Preprint MCS-P387-1093

The Resonance Strategy*

Larry Wos

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

Abstract

Especially in mathematics and in logic, lemmas (basic truths) play a key role for proving theorems. In ring theory, for example, a useful lemma asserts that, for all elements *x*, the product in either order of 0 and *x* is 0; in two-valued sentential (or propositional) calculus, a useful lemma asserts that, for all *x*, *x* implies *x*. Even in algorithm writing and in circuit design, lemmas play a key role: *minus*(*minus*(*x*)) = *x* in the former and NOT(AND(*x*,*y*)) = OR(NOT(*x*),NOT(*y*)) in the latter. Whether the object is to prove a theorem, write an algorithm, or design a circuit, and whether the assignment is given to a person or (preferably) to an automated reasoning program, the judicious use of lemmas often spells the difference between success and failure. In this article, we focus on what might be thought of as a generalization of the concept of lemma, namely, the concept of *resonator*, and on a strategy, the *resonance strategy*, that keys on resonators. For example, where in Boolean groups—those in which the square of every *x* is the identity element *e*—the lemmas *yzyz* = *e* and *yyzz* = *e* are such that neither generalizes the other, the resonator (formula schema) **** = *e*, by using each occurrence of ''star'' to assert the presence of some variable, generalizes and captures (in a manner that is discussed in this article) both lemmas. Note that the cited resonator, if viewed as a lemma with star replaced by some chosen variable, captures neither cited lemma as an instance. Lemmas of a theory are provably ''true'' in the theory and, therefore, can be used to complete an assignment. In contrast, resonators, which capture collections of equations or collections of formulas that may or may not include truths, are used by the resonance strategy to *direct* the search for the information needed for assignment completion. In addition to discussing how one finds useful resonators, we detail various successes, in some of which the resonance strategy played a key role in obtaining a far better proof and in some of which the resonance strategy proved indispensable. The successes are taken from group theory, Robbins algebra, and various logic calculi.

Key words. Automated reasoning, group theory, logic calculi, open questions, OTTER, resonance strategy, Robbins algebra.

1. Genesis of the Resonance Strategy

We at Argonne National Laboratory constantly seek a question or a problem that is currently out of reach of the most powerful automated reasoning program we have in hand, and then commence our attack. To this date, we have always used for such experiments a program we have designed and implemented. Our goal is, on the one hand, to bring the question or problem within reach of our program and, on the other hand and more generally, to formulate a new strategy, inference rule, or methodology that produces the success. We then test the new strategy, inference rule, or methodology on related problems and then on unrelated problems. Many of our contributions to automated reasoning can be traced directly to applying the cited approach; see Chapter 2 of [Wos87] for appropriate

* This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

anecdotes, and see [Wos93a] for recent successes. Indeed, this article typifies our preferred approach to research in automated reasoning, research that continues to extend the Argonne paradigm for the automation of logical reasoning (presented in detail in [Wos92]).

To enable researchers to experience and test the power of the cited paradigm, we include various input files for the study of various areas of mathematics and logic. Each of the input files can be put to a broad use by appropriate modification of its elements, commenting out or commenting in various options. To comment out an element, one simply places a ''%'' before the element to be ignored by McCune's program OTTER [McCune90]. To comment in an option so that it can be used, one simply removes the ''%'' that prevents its use. As a further aid to accessing the material in this lengthy paper, the following map is in order.

Section 1.1 answers the natural question concerning the original motivation for the research reported here, research that culminated in Experiment 0 (presented in Section 1.2) that gave birth to the *resonance strategy* and to the concept of *resonator* (defined formally in Section 2). To provide a comparison with more familiar notions, in Section 1.3 we discuss the concept of resonator as a generalization of the concept of lemma (so frequently used in mathematics). In Section 1.4 we offer 68 theorems (or theses) that serve well for evaluating new ideas, new approaches, and new reasoning programs, and in Section 1.5 we offer some experimental results that preceded the formulation of the resonance strategy. A comparison of these early results with successes obtained with the use of the resonance strategy strongly suggests the power offered by the strategy. To complete the picture, we then turn in Section 2 to the *resonance principle* and (as promised) the formal definition of a *resonator* so central to the resonance strategy.

Because we continually seek to encourage the use of McCune's program OTTER, Section 3 is devoted to various features of this program. There we briefly touch on representation (in Section 3.1) and lists for presenting a question or problem, strategy (in Sections 3.2 and 3.4) for controlling reasoning, and (in Sections 3.3, 3.5, and 3.6) a means (weighting) for imposing the researcher's knowledge and intuition on a program's actions. Of the various inference rules OTTER offers for drawing conclusions, the only one that is featured is *paramodulation* (in Section 3.8), for the actions of this inference rule (more than those of any other rule) are far more complex. Sections 3.7 and 3.9 are devoted, respectively, to other OTTER commands and to source material for learning more about this program.

To provide hard evidence for the value of using the resonance strategy, we turn to its iterative use in Section 4 for the study of two-valued sentential (or propositional calculus) and to its use in Sections 5 and 6, respectively, for the study of Robbins algebra and group theory. These uses of the resonance strategy might be termed *static*, as opposed to its *dynamic* use as discussed in Section 2. As further evidence of the power offered by the resonance strategy, we turn in Section 7 to its use for completing elegant proofs, especially in the context of seeking shorter proofs. There we discuss various aspects of elegance, including that of *compactness*, a property that may never have been the object of previous research. To stimulate research, we also offer (in Section 7.1) a question focusing on shorter proofs—a question that nicely complements that (posed in Section 5) which is still open concerning Robbins algebra—and (in Section 7.2) feature a charming proof (of a deep theorem in many-valued sentential calculus), a proof that avoids the use of terms of a certain type. We conclude the section with a brief discussion (in Section 7.3) of various practical aspects that focus on circuit design and on algorithm synthesis.

1.1. Scott's Challenge

The genesis of the research reported in this article was a set of 68 theorems from two-valued sentential (or propositional) calculus [Lukasiewicz63] suggested by Dana Scott [Scott90]; see Section 1.4 for the 68 theorems, which Lukasiewicz calls theses and which throughout this article we sometimes call theorems and sometimes call theses. An axiomatization for this area of logic consists of the following three formulas in which the function *i* can be interpreted as ''implication'' and the function *n* can be interpreted as ''negation''.

(L1) i(i(x,y),i(i(y,z),i(x,z)))
(L2) i(i(n(x),x),x)
(L3) i(x,i(n(x),y))

An axiom set is complete for two-valued sentential calculus if the set of formulas that can be deduced from it with the use of *condensed detachment* (defined and illustrated shortly) and instantiation is precisely the set of formulas (in effect, unit clauses) that are true under all assignments of true and false, with *implication* and *negation* used as logicians use these terms. Scott (in effect) issued a challenge to our automated reasoning program OTTER [McCune90] by asking how many of the 68 theorems (given in Section 1.4) it could prove, using as axioms *L*1, *L*2, and *L*3; see [Wos91b] for additional detail and related items. In contrast to that earlier paper in which we presented a 30-step proof establishing that the Lukasiewicz axiom system (consisting of theses *L*1, *L*2, and *L*3) implies the Church axiom system (consisting of theses 18, 35 and 49), in Section 7.1 we present shorter proofs—indeed, three different proofs, each of length 22.

With my colleague William McCune (author of OTTER), the challenge was accepted. Of especial interest to us was our *lack* of familiarity with two-valued sentential calculus and with the Lukasiewicz axiom system. From the viewpoint of research, more important than the excitement produced by using our program in a new area was the fact that any successes would clearly be free of the justifiable concern that we might have substantially aided the program, even if unwittingly. As evidence of our lack of knowledge and bias, when we began our attack, we did not even know that the 68 theorems are tightly coupled. Indeed, only after our formulation of the *resonance strategy*, which is the focus of this article, did we learn of the coupling.

To instruct OTTER to attempt to prove the 68 theorems, we were fortunate in that the study of this area of logic often relies on the use of *condensed detachment*, an inference rule that can be applied by a reasoning program by using hyperresolution and the following clause, where ‘‘ | ’’ means logical **or** and ‘‘- means logical **not**. Also, the predicate *P* can be interpreted as ‘‘provable’’ and the function *i* as ‘‘implication’’ (with *i* an encoding of ->).

```
% The following clause is used with hyperresolution for condensed detachment.
-P(i(x,y)) | -P(x) | P(y).
```

For this area of logic, condensed detachment considers two formulas, ($A \rightarrow B$) and *C*—respectively called the *major* and the *minor premiss*—and, if *C unifies* with *A*, yields the formula *D*, where *D* is obtained by applying to *B* the most general unifier of *C* and *A*. (For the curious, and consistent with earlier publications, the word ‘‘premiss’’ is spelled as Church recommends.) Unification is a procedure that considers two expressions and seeks to find the most general substitution of terms for variables that makes the two identical. Unification is the basis of many of the procedures applied by OTTER and, more generally, relied upon in automated reasoning (see [Wos87,Wos92]).

Just for illustration, if one applies condensed detachment to

```
i(x,i(x,i(y,y)))
```

and

```
i(z,z)
```

with the second formula playing the role of *C*, one obtains

```
i(i(z,z),i(y,y)).
```

If one reverses the roles of the two formulas and applies condensed detachment, one obtains a copy of the first formula.

In addition to instructing OTTER to use hyperresolution, we relied on three of the program’s features to begin our attack on the 68 theorems suggested by Scott. First, the researcher can assign a maximum to the number of proofs to be sought in a single run; in our case, we assigned the maximum to be infinity, for any given theorem might be proved more than once, making the assignment of 68 unwise. Second, one can place the denials of theorems under attack in a special list, list(passive), to prevent their playing any role in the reasoning other than ascertaining the completion of a proof. When having a program attempt to prove a number of independent theorems, a wise move is to prevent their interaction. We did place the negation (or denial) of each of the 68 theorems in list(passive). Third, to ascertain which theorems had been proved while the run was in progress, we appended to each denial an ANSWER literal with the corresponding thesis number; Lukasiewicz calls the theorems under discussion *theses*.

Consistent with our view that the use of strategy is crucial in almost all cases—both strategy to restrict a program's reasoning and strategy to direct it—we used the *set of support strategy* and *weighting*. For the set of support strategy, which restricts the reasoning by requiring that all conclusions that are drawn be recursively traceable to a designated input set (the initial set of support), we placed in list(sos) the three Lukasiewicz axioms $L1$, $L2$, and $L3$. Regarding weighting, which permits one to define the complexity of a conclusion to reflect one's knowledge and intuition and thus strongly influence OTTER's choice of where next to focus attention, we simply used symbol count. After all, we had no knowledge of the field and certainly no intuition; therefore, the natural move was to instruct the program to choose as the next focus of attention the simplest available conclusion, where simplicity was measured in terms of symbol count.

Our initial attack was indeed encouraging: In the first run, OTTER proved 37 of the 68 theorems to be proved. However, essentially no additional progress was made until we turned to the use of a parallel version of OTTER (ROO), whose use eventually led to proofs of 48 of the 68 theorems. We could get no further. Although proving 48 out of 68 theorems might be considered as meeting the original Scott challenge moderately well, to be completely candid, we wished to send to Scott a more impressive total success.

1.2. The Resonance Strategy Is Born

More than thirty years of experimentation with automated reasoning programs has strongly suggested that, sometimes, intuition or a guess without evidence should be boldly followed. Such a move is especially appealing since, when compared with research conducted by hand, one of the advantages of relying on the assistance of an automated reasoning program to investigate a wild guess is that the latter costs relatively little in time or energy.

The hypothesis (or guess) that drove the next phase of the attack on the 68 theorems was to conjecture that, since they are all from the same field of logic, perhaps some (possibly strong) connection existed among them. Even if they were not connected, their shape or functional skeleton, ignoring variables, might hold an important clue for directing OTTER's search for the desired 68 proofs. For a taste of what is meant by ''shape'' of a formula, we offer the following two expressions which, though not identical, have the same shape.

```
i(i(x,i(y,z)),i(i(y,x),i(y,z)))
i(i(x,i(y,z)),i(i(x,y),i(x,z)))
```

The two cited formulas are in fact identical *if* one ignores the particular variables or, equivalently, considers the variables to be indistinguishable.

We then placed the clause equivalent of each of the 68 theorems in a list, weight_list(pick_and_purge), that is used by OTTER to direct its search and also to decide which information to purge upon generation; we note that a more complete treatment of the subject is given in Section 3.5. The placement of a formula in this list in no way asserts that the formula holds or is a valid lemma. The clause equivalent (for OTTER) of each of the theorems is obtained by prefixing each with some predicate, say *P*, postfixing each with a period, and (of course) adding an additional opening and an additional closing parenthesis. We also assigned to each of the 68 clauses under discussion a *weight* of 2, causing the program to prefer for the focus of attention *any* retained formula that matched in shape (ignoring the specific variables present) any of the 68 formulas. By setting the maximum of the weight of retained formulas equal to 20, *any* deduced formula that matched in shape one of the 68 formulas under discussion was retained, for it was given a weight of 2. (The max_weight was set to 20 because the longest formula in symbol count among the 68 has length 20, when the predicate is counted.) Both for being chosen as the focus of attention to direct the reasoning and for deciding whether to be retained (rather than discarded), any other formula was given a weight equal to its symbol count.

When OTTER executed the new approach of directing its search by keying on formulas that match in shape one of the 68 theorems to be proved, in a single run, all were proved in less than 16 CPU-minutes on a SPARCstation 1+—and the resonance strategy was born. (We shall refer to this experiment throughout this article as Experiment 0.) We thus had the first evidence of the potential power of using the resonance strategy, a strategy that enables the researcher to designate chosen

formulas or equations to be of especial interest because of their symbol pattern (or shape), with variables ignored. However, when one learns as we later did that the 68 theses are in fact tightly coupled, one might have an immediate and justifiable concern that the program is (in effect) proof checking some larger result. This concern was expressed by one of my colleagues early in the study of this new resonance strategy.

In Sections 4-7, we provide ample evidence to put this concern to rest. As part of that evidence, we discuss the case in which the use of resonators—implemented with weight templates to direct the program's search—that were taken from a theorem from Robbins algebra led to the discovery of a shorter proof of that same theorem. This success motivated a number of studies (cited in Sections 6 and 7) featuring the resonance strategy to seek proofs more elegant than previously known, studies that culminated in finding far shorter proofs in other areas that include group theory and logic calculi.

1.3. Resonator as Generalization of Lemma

That proof checking is not what is occurring is reinforced when one recalls that resonators are more general than are lemmas, for lemmas assert truths whereas resonators assert nothing. Resonators instead capture a set of statements, some of which may in fact be false. To produce the set of statements captured by a given resonator *R*, one begins by noting that a resonator is a formula schema obtained from a formula or equation by replacing each variable occurrence in the formula or equation by *; the formal definition is given in Section 2. The statements captured by *R* are precisely those that can be obtained by replacing each occurrence of * by a variable, with no constraint placed on whether the variables must be distinct; indeed, the various occurrences of * in *R* can be assigned the same or different variable names. For example, where in Boolean groups—those in which the square of every *x* is the identity element *e*—the lemmas *yzyz* = *e* and *yyzz* = *e* are such that neither generalizes the other, the resonator (formula schema) **** = *e*, by using each occurrence of ''star'' to assert the presence of some variable, generalizes and captures (in a manner that is discussed in this article) both lemmas. In contrast, among the equations captured by the resonator **** = *e* is the equation *xxxy* = *e*, which is clearly not true in all Boolean groups.

Expressed in one of the acceptable ways to OTTER, the inclusion of the following resonator (where the function *f* denotes product and the occurrences of * have been replaced by arbitrarily chosen variable names) will ordinarily cause OTTER to prefer for the focus of attention equations such as *yzyz* = *e* and *yyzz* = *e*.

weight(f(x,f(y,f(z,x))) = e,2).

The point here is that OTTER treats all variables in a weight template (of the type just given) as indistinguishable or ignorable, other than recognizing them as variables. Of course, neither cited equation that might ordinarily be preferred will be retained, when deduced, if forward subsumption (defined in Section 3.7) is in use and the hypothesis *xx* = *e* that defines Boolean groups is present.

For a second example, let us consider some of the consequences of including the following resonator (expressed in OTTER notation), which corresponds to thesis 35 as numbered by Lukasiewicz. (Throughout this article, when we talk about a resonator that corresponds to some thesis or the like, we mean that it has the same shape or functional skeleton as does the thesis and that the particular variables which replace the occurrences of star are irrelevant. Further, we typically talk about the variables in a resonator, recognizing that, in a strict sense, the occurrences of star designate variable presence in that position.) For clarity, although the following expression is (as a formula presented in clause form) identical to thesis 35, as a resonator (in OTTER notation), the variables are ignored.

weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),2).

Because a weight of 2 has been assigned to the template, any formula—in particular, the following two—that matches it in shape will be given preferential treatment for directing the program's reasoning.

i(i(x,i(y,z)),i(i(y,x),i(y,z)))
i(i(x,i(y,z)),i(i(x,y),i(x,z)))

To mechanically test a formula or equation for possibly ''matching'' a resonator expressed in the star notation, one first replaces all occurrences of star with variables such that no two variables are identical, and then one determines whether a renaming of the variables yields the formula or equation; if such a

renaming exists, then the formula or equation matches a resonator.

Because variables in a resonator (expressed in OTTER notation) are considered to be indistinguishable, both formulas are ‘‘captured’’ by the given resonator. In contrast, if one considers the corresponding clauses or (possible) lemmas, then the second of the two formulas, which itself is (in effect) identical to the given resonator, is *not* captured or generalized by the first, nor is the first captured by the second. We again see how the concept of resonator is (in the illustrated sense) a generalization of the concept of lemma. We also see that resonators can be used to *guide* a reasoning program’s search, a use that contrasts nicely with the use of lemmas as truths from which to *reason*. If *no* statements that are true in the theory under study are captured by a given resonator, then, most likely, the inclusion of the resonator is pointless, for no deduced conclusion will match the resonator. On the other hand, if but a single deduced and retained conclusion is captured by a resonator—and hence given preference dictated by the weight assigned to the resonator—the inclusion of that resonator can dramatically improve the effectiveness of the program.

As contrasted with what one might call the syntactic sense, the concept of resonator also generalizes the notion of lemma in the following semantic sense. One distinction sometimes made between the concept of lemma and that of theorem is that the former is often *not* of great interest in and of itself but, instead, is of interest for proving one or more theorems that *are* of interest in and of themselves. Resonators, by being used to increase the effectiveness of a program’s search, are also primarily of interest for their value in proving one or more theorems. Since a single resonator can capture (in the sense just discussed) a large set of lemmas, the concept of resonator generalizes that of lemma. Further, all consequences of a lemma obtained by substitution are true, for a lemma (to be accepted for a theory) must be true. In contrast, and as part of the generalization, a resonator may be such that some statements captured by it are false and some are true.

1.4. The 68 Theorems for Study

For those who might wish to pursue appropriate research, the 68 theses, numbered according to the way Lukasiewicz numbered them, are the following; in particular, theses 1, 2, and 3 are, respectively, *L*1, *L*2, and *L*3.

(thesis_4) i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))
(thesis_5) i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))
(thesis_6) i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))
(thesis_7) i(i(x,i(i(y,z),u)),i(i(y,v),i(x,i(i(v,z),u))))
(thesis_8) i(i(x,y),i(i(z,x),i(i(y,u),i(z,u))))
(thesis_9) i(i(i(n(x),y),z),i(x,z))
(thesis_10) i(x,i(i(i(n(x),x),x),i(i(y,x),x)))
(thesis_11) i(i(x,i(i(n(y),y),y)),i(i(n(y),y),y))
(thesis_12) i(x,i(i(n(y),y),y))
(thesis_13) i(i(n(x),y),i(z,i(i(y,x),x)))
(thesis_14) i(i(i(x,i(i(y,z),z)),u),i(i(n(z),y),u))
(thesis_15) i(i(n(x),y),i(i(y,x),x))
(thesis_16) i(x,x)
(thesis_17) i(x,i(i(y,x),x))
(thesis_18) i(x,i(y,x))
(thesis_19) i(i(i(x,y),z),i(y,z))
(thesis_20) i(x,i(i(x,y),y))
(thesis_21) i(i(x,i(y,z)),i(y,i(x,z)))
(thesis_22) i(i(x,y),i(i(z,x),i(z,y)))
(thesis_23) i(i(i(x,i(y,z)),u),i(i(y,i(x,z)),u))
(thesis_24) i(i(i(x,y),x),x)
(thesis_25) i(i(i(x,y),z),i(i(x,u),i(i(u,y),z)))
(thesis_26) i(i(i(x,y),z),i(i(z,x),x))
(thesis_27) i(i(i(x,y),y),i(i(y,x),x))
(thesis_28) i(i(i(i(x,y),y),z),i(i(i(y,u),x),z))
(thesis_29) i(i(i(x,y),z),i(i(x,z),z))

(thesis_30) i(i(x,i(x,y)),i(x,y))
(thesis_31) i(i(x,y),i(i(i(x,z),u),i(i(y,u),u)))
(thesis_32) i(i(i(x,y),z),i(i(x,u),i(i(u,z),z)))
(thesis_33) i(i(x,y),i(i(y,i(z,i(x,u))),i(z,i(x,u))))
(thesis_34) i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u))))
(thesis_35) i(i(x,i(y,z)),i(i(x,y),i(x,z)))
(thesis_36) i(n(x),i(x,y))
(thesis_37) i(i(i(x,y),z),i(n(x),z))
(thesis_38) i(i(x,n(x)),n(x))
(thesis_39) i(n(n(x)),x)
(thesis_40) i(x,n(n(x)))
(thesis_41) i(i(x,y),i(n(n(x)),y))
(thesis_42) i(i(i(n(n(x)),y),z),i(i(x,y),z))
(thesis_43) i(i(x,y),i(i(y,n(x)),n(x)))
(thesis_44) i(i(x,i(y,n(z))),i(i(z,y),i(x,n(z))))
(thesis_45) i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))
(thesis_46) i(i(x,y),i(n(y),n(x)))
(thesis_47) i(i(x,n(y)),i(y,n(x)))
(thesis_48) i(i(n(x),y),i(n(y),x))
(thesis_49) i(i(n(x),n(y)),i(y,x))
(thesis_50) i(i(i(n(x),y),z),i(i(n(y),x),z))
(thesis_51) i(i(x,i(y,z)),i(x,i(n(z),n(y))))
(thesis_52) i(i(x,i(y,n(z))),i(x,i(z,n(y))))
(thesis_53) i(i(n(x),y),i(i(x,y),y))
(thesis_54) i(i(x,y),i(i(n(x),y),y))
(thesis_55) i(i(x,y),i(i(x,n(y)),n(x)))
(thesis_56) i(i(i(i(x,y),y),z),i(i(n(x),y),z))
(thesis_57) i(i(n(x),y),i(i(x,z),i(i(z,y),y)))
(thesis_58) i(i(i(i(x,y),i(i(y,z),z)),u),i(i(n(x),z),u))
(thesis_59) i(i(n(x),y),i(i(z,y),i(i(x,z),y)))
(thesis_60) i(i(x,i(n(y),z)),i(x,i(i(u,z),i(i(y,u),z))))
(thesis_61) i(i(x,y),i(i(z,y),i(i(n(x),z),y)))
(thesis_62) i(i(n(n(x)),y),i(x,y))
(thesis_63) i(x,i(y,y))
(thesis_64) i(n(i(x,x)),y)
(thesis_65) i(i(n(x),n(i(y,y))),x)
(thesis_66) i(n(i(x,y)),x)
(thesis_67) i(n(i(x,y)),n(y))
(thesis_68) i(n(i(x,n(y))),y)
(thesis_69) i(x,i(n(y),n(i(x,y))))
(thesis_70) i(x,i(y,n(i(x,n(y)))))
(thesis_71) n(i(i(x,x),n(i(y,y))))

1.5. Preliminary Experimental Results

It seems clear that virtually any data that strongly promotes serious experimentation in automated reasoning merits inclusion in an article of this type. In that regard, the following is a list of the 37 theses (by number) that were proved in a single OTTER run *before* the resonance strategy was formulated, the run cited in Section 1.1.

04 05 06 09 10 11 12 13 15 16 17 18 19 20 21 22 23 24
30 36 37 38 39 40 41 42 48 49 50 62 63 64 65 66 67 68 71

The cited theses were *not* proved in the order given. For thesis 10, the program found three proofs. When the original attack was revisited on a SPARCstation 2 (during the writing of this article), thesis 50 was the last of the 37 that was proved, requiring approximately 287 CPU-seconds and completing upon retention of a clause numbered 47404. The run was terminated when the assigned limit of 24

megabytes of memory was reached.

When the resonance strategy was then used by keying on the 68 theses to be proved, and when all other parameters were kept the same, OTTER proved all 68 theses in a single run (see Experiment 0 cited in Section 1.2). When the experiment was revisited on a SPARCstation 2 (during the writing of this article), the last thesis to be proved was thesis 71 in approximately 397 CPU-seconds after retention of a clause numbered 32257. Among the sharp differences with and without the resonance strategy, thesis 71 was proved without the use of the strategy in approximately 69 CPU-seconds after retention of a clause numbered 7689. The fact that the use of the resonance strategy *sometimes* costs CPU time is expected, for the search space is dramatically perturbed by assigning a low weight (high priority) to formulas that match an included resonator. For but one example, thesis 7 would ordinarily be assigned a weight of 20, if symbol count was the rule for priority assignment. With the resonance strategy, thesis 10 was proved six times, and thesis 17 was proved twice.

To further encourage research with the program OTTER, we include the following input file for the study of two-valued sentential calculus. In Section 3, we focus on the use and meaning of the commands found in this file. For the researcher who wishes to avoid using the resonance strategy, it suffices to place a ''%'' before each of the 68 templates found in the weigh_list(pick_and_purge). Otherwise, if the input file is used as given, the resulting experiment will duplicate our use of the resonance strategy in the cited context. (In the following file, '' | '' denotes logical **or**, ''-'' denotes logical **not**, and, with the exception of the clauses that mention Scott, the clauses that mention a person's name correspond to known axiom systems for two-valued sentential calculus; those that mention Scott were included in order to answer questions he posed.)

Input File for Studying Two-Valued Sentential Calculus

```
set(hyper_res).
assign(max_weight,20).
assign(max_proofs, 0).
clear(print_kept).
% clear(for_sub).
clear(back_sub).
% assign(max_seconds, 1200).
assign(max_mem, 24000).
assign(report, 900).
% assign(max_distinct_vars, 4).
% assign(pick_given_ratio, 3).
set(order_history).
set(input_sos_first).
% set(sos_queue).
set(print_level).

weight_list(pick_and_purge).
%  Following are templates corresponding to the 68 theses to be proved.
weight(P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))),2).
weight(P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))),2).
weight(P(i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))),2).
weight(P(i(i(x,i(i(y,z),u)),i(i(y,v),i(x,i(i(v,z),u))))),2).
weight(P(i(i(x,y),i(i(z,x),i(i(y,u),i(z,u))))),2).
weight(P(i(i(i(n(x),y),z),i(x,z))),2).
weight(P(i(x,i(i(i(n(x),x),x),i(i(y,x),x)))),2).
weight(P(i(i(x,i(i(n(y),y),y)),i(i(n(y),y),y)))),2).
weight(P(i(x,i(i(n(y),y),y)))),2).
weight(P(i(i(n(x),y),i(z,i(i(y,x),x)))),2).
weight(P(i(i(i(x,i(i(y,z),z)),u),i(i(n(z),y),u))),2).
```

weight(P(i(i(n(x),y),i(i(y,x),x))),2).
weight(P(i(x,x)),2).
weight(P(i(x,i(i(y,x),x))),2).
weight(P(i(x,i(y,x))),2).
weight(P(i(i(i(x,y),z),i(y,z))),2).
weight(P(i(x,i(i(x,y),y))),2).
weight(P(i(i(i(x,i(y,z)),i(y,i(x,z)))),2).
weight(P(i(i(x,y),i(i(z,x),i(z,y)))),2).
weight(P(i(i(i(x,i(y,z)),u),i(i(y,i(x,z)),u))),2).
weight(P(i(i(i(x,y),x),x)),2).
weight(P(i(i(i(x,y),z),i(i(x,u),i(i(u,y),z)))),2).
weight(P(i(i(i(x,y),z),i(i(z,x),x))),2).
weight(P(i(i(i(x,y),y),i(i(y,x),x))),2).
weight(P(i(i(i(i(x,y),y),z),i(i(i(y,u),x),z))),2).
weight(P(i(i(i(x,y),z),i(i(x,z),z))),2).
weight(P(i(i(x,i(x,y)),i(x,y))),2).
weight(P(i(i(x,y),i(i(i(x,z),u),i(i(y,u),u)))),2).
weight(P(i(i(i(x,y),z),i(i(x,u),i(i(u,z),z)))),2).
weight(P(i(i(x,y),i(i(y,i(z,i(x,u))),i(z,i(x,u))))),2).
weight(P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u))))),2).
weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),2).
weight(P(i(n(x),i(x,y))),2).
weight(P(i(i(i(x,y),z),i(n(x),z))),2).
weight(P(i(i(x,n(x)),n(x))),2).
weight(P(i(n(n(x)),x)),2).
weight(P(i(x,n(n(x)))),2).
weight(P(i(i(x,y),i(n(n(x)),y))),2).
weight(P(i(i(i(n(n(x)),y),z),i(i(x,y),z))),2).
weight(P(i(i(x,y),i(i(y,n(x)),n(x)))),2).
weight(P(i(i(x,i(y,n(z))),i(i(z,y),i(x,n(z))))),2).
weight(P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))),2).
weight(P(i(i(x,y),i(n(y),n(x)))),2).
weight(P(i(i(x,n(y)),i(y,n(x)))),2).
weight(P(i(i(n(x),y),i(n(y),x))),2).
weight(P(i(i(n(x),n(y)),i(y,x))),2).
weight(P(i(i(i(n(x),y),z),i(i(n(y),x),z))),2).
weight(P(i(i(x,i(y,z)),i(x,i(n(z),n(y))))),2).
weight(P(i(i(x,i(y,n(z))),i(x,i(z,n(y))))),2).
weight(P(i(i(n(x),y),i(i(x,y),y))),2).
weight(P(i(i(x,y),i(i(n(x),y),y))),2).
weight(P(i(i(x,y),i(i(x,n(y)),n(x)))),2).
weight(P(i(i(i(i(x,y),y),z),i(i(n(x),y),z))),2).
weight(P(i(i(n(x),y),i(i(x,z),i(i(z,y),y)))),2).
weight(P(i(i(i(i(x,y),i(i(y,z),z)),u),i(i(n(x),z),u))),2).
weight(P(i(i(n(x),y),i(i(z,y),i(i(x,z),y)))),2).
weight(P(i(i(x,i(n(y),z)),i(x,i(i(u,z),i(i(y,u),z))))),2).
weight(P(i(i(x,y),i(i(z,y),i(i(n(x),z),y)))),2).
weight(P(i(i(n(n(x)),y),i(x,y))),2).
weight(P(i(x,i(y,y))),2).
weight(P(i(n(i(x,x)),y)),2).
weight(P(i(i(n(x),n(i(y,y))),x)),2).
weight(P(i(n(i(x,y)),x)),2).
weight(P(i(n(i(x,y)),n(y))),2).
weight(P(i(n(i(x,n(y))),y)),2).
weight(P(i(x,i(n(y),n(i(x,y))))),2).

```
weight(P(i(x,i(y,n(i(x,n(y))))))),2).
weight(P(n(i(i(x,x),n(i(y,y))))),2).
end_of_list.

list(usable).
%  The following clause is used with hyperresolution for condensed detachment.
-P(i(x,y)) | -P(x) | P(y).
%  The following disjunctions, except those mentioning Scott,
%  are the negation of known axiom systems.

-P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | -P(i(n(n(p)),p)) |
-P(i(p,n(n(p)))) | -P(i(i(p,q),i(n(q),n(p)))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
$ANSWER(step_allFrege_18_35_39_40_46_21).  % 21 is dependent.

-P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
-P(i(i(q,r),i(i(p,q),i(p,r)))) | -P(i(p,i(n(p),q))) |
-P(i(i(p,q),i(i(n(p),q),q))) | -P(i(i(p,i(p,q)),i(p,q))) |
$ANSWER(step_allHilbert_18_21_22_3_54_30).  % 30 is dependent.

-P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) |
-P(i(i(n(p),n(q)),i(q,p))) | $ANSWER(step_allBEH_Church_FL_18_35_49).

-P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) |
-P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) | $ANSWER(step_allLuka_x_19_37_59).

-P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) |
-P(i(i(s,i(n(p),r)),i(s,i(i(q,r),i(i(p,q),r))))) | $ANSWER(step_allWos_x_19_37_60).

-P(i(i(p,q),i(i(q,r),i(p,r)))) | -P(i(i(n(p),p),p)) |
-P(i(p,i(n(p),q))) | $ANSWER(step_allLuka_1_2_3).

-P(i(p,p)) | -P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
-P(i(i(i(p,q),p),p)) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) |
-P(i(n(n(p)),p)) | -P(i(p,n(n(p)))) | -P(i(i(p,q),i(n(q),n(p)))) |
$ANSWER(step_allScott_orig_16_18_21_24_35_39_40_46).

-P(i(p,p)) | -P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
-P(i(i(i(p,q),p),p)) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) |
-P(i(n(n(p)),p)) | -P(i(p,n(n(p)))) | -P(i(i(n(p),n(q)),i(q,p))) |
$ANSWER(step_allScott_orig0_16_18_21_24_35_39_40_49).

end_of_list.

list(sos).
%  The following three are Luka, 1 2 3.
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(n(x),x),x)).
P(i(x,i(n(x),y))).

%  The following are from Frege, 18 35 21 46 39 40, with 21 dependent.
% P(i(x,i(y,x))).  %  axiom F1.
% P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).  %  axiom F2.
% P(i(i(x,i(y,z)),i(y,i(x,z)))).  %  axiom F3.
% P(i(i(x,y),i(n(y),n(x)))).  %  axiom F4.
% P(i(n(n(x)),x)).  %  axiom F5.
```

```
% P(i(x,n(n(x)))).  %  axiom f6.
end_of_list.

list(passive).
-P(i(i(i(i(q,r),i(p,r)),s),i(i(p,q),s))) | $ANS(neg_th_04).
-P(i(i(p,i(q,r)),i(i(s,q),i(p,i(s,r))))) | $ANS(neg_th_05).
-P(i(i(p,q),i(i(i(p,r),s),i(i(q,r),s)))) | $ANS(neg_th_06).
-P(i(i(t,i(i(p,r),s)),i(i(p,q),i(t,i(i(q,r),s))))) | $ANS(neg_th_07).
-P(i(i(q,r),i(i(p,q),i(i(r,s),i(p,s))))) | $ANS(neg_th_08).
-P(i(i(i(n(p),q),r),i(p,r))) | $ANS(neg_th_09).
-P(i(p,i(i(i(n(p),p),p),i(i(q,p),p)))) | $ANS(neg_th_10).
-P(i(i(q,i(i(n(p),p),p)),i(i(n(p),p),p))) | $ANS(neg_th_11).
-P(i(t,i(i(n(p),p),p))) | $ANS(neg_th_12).
-P(i(i(n(p),q),i(t,i(i(q,p),p)))) | $ANS(neg_th_13).
-P(i(i(i(t,i(i(q,p),p)),r),i(i(n(p),q),r))) | $ANS(neg_th_14).
-P(i(i(n(p),q),i(i(q,p),p))) | $ANS(neg_th_15).
-P(i(p,p)) | $ANS(neg_th_16).
-P(i(p,i(i(q,p),p))) | $ANS(neg_th_17).
-P(i(q,i(p,q))) | $ANS(neg_th_18).
-P(i(i(i(p,q),r),i(q,r))) | $ANS(neg_th_19).
-P(i(p,i(i(p,q),q))) | $ANS(neg_th_20).
-P(i(i(p,i(q,r)),i(q,i(p,r)))) | $ANS(neg_th_21).
-P(i(i(q,r),i(i(p,q),i(p,r)))) | $ANS(neg_th_22).
-P(i(i(i(q,i(p,r)),s),i(i(p,i(q,r)),s))) | $ANS(neg_th_23).
-P(i(i(i(p,q),p),p)) | $ANS(neg_th_24).
-P(i(i(i(p,r),s),i(i(p,q),i(i(q,r),s)))) | $ANS(neg_th_25).
-P(i(i(i(p,q),r),i(i(r,p),p))) | $ANS(neg_th_26).
-P(i(i(i(p,q),q),i(i(q,p),p))) | $ANS(neg_th_27).
-P(i(i(i(i(r,p),p),s),i(i(i(p,q),r),s))) | $ANS(neg_th_28).
-P(i(i(i(p,q),r),i(i(p,r),r))) | $ANS(neg_th_29).
-P(i(i(p,i(p,q)),i(p,q))) | $ANS(neg_th_30).
-P(i(i(p,s),i(i(i(p,q),r),i(i(s,r),r)))) | $ANS(neg_th_31).
-P(i(i(i(p,q),r),i(i(p,s),i(i(s,r),r)))) | $ANS(neg_th_32).
-P(i(i(p,s),i(i(s,i(q,i(p,r))),i(q,i(p,r))))) | $ANS(neg_th_33).
-P(i(i(s,i(q,i(p,r))),i(i(p,s),i(q,i(p,r))))) | $ANS(neg_th_34).
-P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | $ANS(neg_th_35).
-P(i(n(p),i(p,q))) | $ANS(neg_th_36).
-P(i(i(i(p,q),r),i(n(p),r))) | $ANS(neg_th_37).
-P(i(i(p,n(p)),n(p))) | $ANS(neg_th_38).
-P(i(n(n(p)),p)) | $ANS(neg_th_39).
-P(i(p,n(n(p)))) | $ANS(neg_th_40).
-P(i(i(p,q),i(n(n(p)),q))) | $ANS(neg_th_41).
-P(i(i(i(n(n(p)),q),r),i(i(p,q),r))) | $ANS(neg_th_42).
-P(i(i(p,q),i(i(q,n(p)),n(p)))) | $ANS(neg_th_43).
-P(i(i(s,i(q,n(p))),i(i(p,q),i(s,n(p))))) | $ANS(neg_th_44).
-P(i(i(s,i(q,p)),i(i(n(p),q),i(s,p)))) | $ANS(neg_th_45).
-P(i(i(p,q),i(n(q),n(p)))) | $ANS(neg_th_46).
-P(i(i(p,n(q)),i(q,n(p)))) | $ANS(neg_th_47).
-P(i(i(n(p),q),i(n(q),p))) | $ANS(neg_th_48).
-P(i(i(n(p),n(q)),i(q,p))) | $ANS(neg_th_49).
-P(i(i(i(n(q),p),r),i(i(n(p),q),r))) | $ANS(neg_th_50).
-P(i(i(p,i(q,r)),i(p,i(n(r),n(q))))) | $ANS(neg_th_51).
-P(i(i(p,i(q,n(r))),i(p,i(r,n(q))))) | $ANS(neg_th_52).
-P(i(i(n(p),q),i(i(p,q),q))) | $ANS(neg_th_53).
-P(i(i(p,q),i(i(n(p),q),q))) | $ANS(neg_th_54).
```

```
-P(i(i(p,q),i(i(p,n(q)),n(p)))) | $ANS(neg_th_55).
-P(i(i(i(i(p,q),q),r),i(i(n(p),q),r))) | $ANS(neg_th_56).
-P(i(i(n(p),r),i(i(p,q),i(i(q,r),r)))) | $ANS(neg_th_57).
-P(i(i(i(i(p,q),i(i(q,r),r)),s),i(i(n(p),r),s))) | $ANS(neg_th_58).
-P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) | $ANS(neg_th_59).
-P(i(i(s,i(n(p),r)),i(s,i(i(q,r),i(i(p,q),r))))) | $ANS(neg_th_60).
-P(i(i(p,r),i(i(q,r),i(i(n(p),q),r)))) | $ANS(neg_th_61).
-P(i(i(n(n(p)),q),i(p,q))) | $ANS(neg_th_62).
-P(i(q,i(p,p))) | $ANS(neg_th_63).
-P(i(n(i(p,p)),q)) | $ANS(neg_th_64).
-P(i(i(n(q),n(i(p,p))),q)) | $ANS(neg_th_65).
-P(i(n(i(p,q)),p)) | $ANS(neg_th_66).
-P(i(n(i(p,q)),n(q))) | $ANS(neg_th_67).
-P(i(n(i(p,n(q))),q)) | $ANS(neg_th_68).
-P(i(p,i(n(q),n(i(p,q))))) | $ANS(neg_th_69).
-P(i(p,i(q,n(i(p,n(q)))))) | $ANS(neg_th_70).
-P(n(i(i(p,p),n(i(q,q))))) | $ANS(neg_th_71).
end_of_list.

% list(demodulators).
%  (n(n(x)) = junk).
%  (n(n(n(x))) = junk).
%  (i(i(x,x),y) = junk).
%  (i(y,i(x,x)) = junk).
% (i(junk,x) = junk).
% (i(x,junk) = junk).
% (n(junk) = junk).
% (P(junk) = $T).
% end_of_list.
```

2. The Resonance Principle

As with many principles, the *resonance principle* is more a strong suggestion than a hard-and-fast rule. The principle asserts that formulas or equations found in proofs of theorems of a field provide useful resonators (formally defined shortly) for attempting to prove theorems from the same field. Captured as a special case is that of using as resonators steps of a proof of theorem **T** to prove **T**. Perhaps unexpected—it was for us when first tried—the special case of using the resonance strategy (defined in this section) does *not necessarily* simply reproduce the known proof but, instead, often produces a far more elegant proof (see Sections 4, 5, 6, and 7 for some marked successes in that regard).

Definition. A *resonator R* is a formula schema obtained from a formula or equation by replacing each variable occurrence in the formula or equation by *. The resonator *R* implicitly represents the set of formulas or equations obtained by assigning a variable name to each occurrence of * in *R*; the represented set may include formulas or equations that are false in the theory under study. The various occurrences of * in *R* can be assigned the same or different variable names; no restriction on the assignment exists. To each resonator *R* is attached an integer *w*, called its *value* or *weight*, that reflects the conjectured importance of *R*. The smaller the value or weight of a resonator, the higher is its conjectured importance. Implicit in the definition of resonator is the role the researcher plays, for, typically, it is the researcher who chooses the resonators and who assigns the value or weight to each resonator.

Because, when expressed in OTTER notation (with * replaced by a variable), variables in a resonator are treated as indistinguishable and ignorable, a single resonator can capture a large set of statements, possibly including items that are false in the theory of mathematics or logic under study. (When we talk about the variables in a resonator, we recognize that, in a strict sense, the occurrences of star *designate* variable presence in that position.) For example, in Boolean groups, the resonator (expressed in OTTER notation) *xyzx = e* (the identity element) captures the equations *yzyz = e* and *yyzz = e*, both of which are (true) lemmas such that neither generalizes the other. The given resonator also captures

the equation *xxxy = e*, which is false in most Boolean groups. A Boolean group is a group in which the square of every element *x* is the identity *e*. Because variables are ignored (when OTTER examines them or examines any weight template), nothing different occurs if the given resonator is replaced by the resonator *xxxx = e* or by the resonator *xyzu = e*. As discussed in some detail in Section 1.3, the concept of resonator is more general than is the concept of lemma in the sense that one resonator can capture many lemmas. Where a lemma in a theory is ''true'' in the theory, a resonator for a theory is not intended to take on the value **true** or **false**.

Definition. The *static resonance strategy*, often referred to simply as the *resonance strategy*, is a direction strategy that relies on the use of resonators supplied by the researcher.

Indeed, rather than being a restriction strategy for constraining a program's reasoning, the resonance strategy is a direction strategy for steering a program's reasoning by influencing it in its choice of where next to focus its attention. In this article, we present the results of OTTER's use of the resonance strategy in studies focusing on areas of mathematics and logic including group theory, Robbins algebra, and various logic calculi. We have also formulated and experimented briefly with a companion to the resonance strategy. The companion strategy, called the *resonance-restriction* strategy, *restricts* a program's reasoning with resonators rather than *directing* its reasoning with resonators. The strategy rests on using the steps of a known proof to filter the set of deduced conclusions, retaining only those that (with variables ignored) match in shape one of the steps, but directing the search by symbol-count preference rather than by resonance. (A mechanical test for matching is given in Section 1.3.)

In the future, in contrast to the static resonance strategy, we plan to investigate the case in which resonators are adjoined *during* a run, believing that the resulting *dynamic resonance strategy* might prove to be an extremely powerful strategy. One incarnation of the dynamic resonance strategy would add as resonators the proof steps of so-called lesser theorems of interest proved during the attack on a so-called greater theorem of interest, where the lesser theorems were included to enable the program to find possibly useful resonators. Even without access to an implementation of the dynamic resonance strategy, one can (as we have in many experiments) simulate its use; see Section 4. One begins with simple theorems and has OTTER obtain proofs of as many of them as the allotted CPU time and memory permit. From each completed proof, one then takes the proof steps, excluding those found in the input as axioms and the like, and uses them as resonators in the attempt to prove harder theorems. Further, as we show in Section 4, one can profitably iterate in the cited manner. The two-step and the iterated approach can each produce a valuable set of resonators for studying various aspects of the theory from which the selected theorems are taken, and it can at the same time produce valuable resonators for applying the resonance strategy in the pursuit of the proof of a very deep, purported theorem.

Although in the context of the dynamic resonance strategy we have at this time little experimental evidence concerning the effectiveness of the cited two-step approach for accruing powerful resonators, in the context of the static resonance strategy we have much. In particular, we have used with great success a closely related variant of the approach for accruing resonators where the goal was that of seeking more elegant proofs. In one study, we applied an iterative approach that began with a 33-step proof [Lukasiewicz63} of the completeness (for two-valued sentential calculus) of the axiom system consisting of *L*1, *L*2, and *L*3 (given in Section 1.1). (For computing the length of a proof, we ignore the axioms and count just the deduced steps; condensed detachment, defined and illustrated in Section 1.1, is the inference rule that is used in this area of logic.) The Lukasiewicz proof completes by deriving Church's axiom system consisting of theses 18, 35, and 49 (given in Section 1.4). With heavy use of the resonance strategy, we eventually found various 22-step proofs of the Church axiom system using condensed detachment; see Section 7.1. Whether a shorter proof exists is at this time unknown and is an interesting and challenging research question.

3. A Brief Introduction to the Program OTTER

To aid the researcher in understanding more fully the successes obtained with McCune's program OTTER and to provide a small beginning for one who wishes to use this program, we now give a brief introduction to some of its features. A thorough treatment is provided in manual form in [McCune90] and in book form in [Wos92], the latter offering in Chapter 16 a tutorial for the use of OTTER. The following discussion will also add to one's understanding of the items found in the input file given in

Section 1.5 and the input files given later in Sections 5, 6, and 7.2.

3.1. Problem Representation and Clause Lists

Although OTTER accepts a number of notations, including first-order predicate calculus, we generally submit a question or problem to the program in the clause language. Examples of clauses are given in the input file of Section 1.5; recall that ‘‘ | ’’ denotes logical **or**, ‘‘-’’ denotes logical **and**, and all variables (such as *x*, *y*, *z*, and *u*) are implicitly universally quantified (meaning, for example, for all *x*).

The clauses used to convey facts, relationships, equations, the assumed falseness of the conclusion of a theorem to be proved, and such can be placed on any of four lists. We place in list(sos) the clauses that the program (initially) selects from to drive its reasoning, and place in list(usable) clauses that can *never* be used in this manner. Clauses in list(usable) are used only to complete the application of an inference rule. For example, the following clause (for condensed detachment) is placed in list(usable), for the intention is that it never drive the reasoning.

-P(i(x,y)) | -P(x) | P(y).

If one wishes OTTER to make a bidirectional search, reasoning forward from some information and backward from the assumed falseness of the theorem under study, then one places in list(sos) some or all of the clauses that convey the nature of the theory under study and the hypothesis of the theorem and some or all of the clauses that correspond to assuming the theorem false. However, as evidenced in the input file of Section 1.5, if one intends that the clauses that correspond to assuming the theorem false play no active role in drawing conclusions, then they are placed in list(passive). As one sees, in our initial study of two-valued sentential calculus prompted by Scott’s e-mail, we placed the negated (or denial) form of theses 4 through 71 in list(passive). The last of the four lists, list(demodulators), for receiving input clauses is typically used for rules (equations) for rewriting information into some desired canonical form.

3.2. Restriction Strategy and Direction Strategy

As the presence of the four lists suggests, especially list(sos) to the researcher familiar with automated reasoning, OTTER offers the use of strategy. We have always maintained that, without the use of strategy, deep questions and difficult problems are in almost all cases out of reach of a reasoning program. Crucial is the use of two types of strategy, strategy that *restricts* the program’s reasoning and strategy that *directs* it. Further, we maintain that, within these two categories of strategy, tough assignments require for their completion a variety of each type. These views, supported by thirty years of experimentation, in part explain our lack of enthusiasm for studying mathematics and logic with a paradigm offering little or no strategy, for example, a paradigm such as that based on logic programming.

Regarding strategy that restricts reasoning, as suggested by the name list(sos), we strongly prefer the use of the set of support strategy [Wos65,Wos87,Wos92]. Intuitively, the set of support strategy requires one to choose a nonempty subset of the input clauses and restrict the reasoning to lines of inquiry that begin with a clause in the chosen subset. To use the set of support strategy, one partitions the set *S* of input clauses into two subsets, a nonempty subset *T* (called the initial set of support) and *S* – *T*; the latter may be empty. The set of support strategy restricts the application of an inference rule by disallowing application of the rule to sets of clauses consisting solely of clauses from the set *S* – *T*. Put another way, all newly retained clauses are added to *T* (the set of support), and each set of clauses to which an inference rule is applied must contain at least one clause in *T*. When the axioms of a theory are placed in list(usable) and not in list(sos), the program is prevented from exploring the theory as a whole and, instead, is more or less confined to focusing on the specific theorem under study. To answer an immediate and natural question arising from a glance at the input file in Section 1.5, we sometimes place axioms in list(sos), especially when we have little else to reason from.

3.3. Weighting

Even with an effective restriction strategy in use, a study benefits markedly from the use of one or more direction strategies. To direct a program’s reasoning, at one end of the spectrum, one can use

weighting [McCharen76,Wos87,Wos92], and at the other end of the spectrum, one can use *level saturation* (equivalently, breadth first). Weighting can be used to convey the researcher's notion regarding complexity preference. Where my colleague R. Overbeek, who formulated weighting, considers one of its uses to be that of circumscribing the terms that the researcher conjectures to play the important role in an assignment, we consider that function of weighting to be one of advice giving; weighting can also be used to discard deduced conclusions that are thought to be of no use. Indeed, we have always thought the object of including templates expressing preference of one function over another or of one combination of functions over another was to reflect intuition or knowledge. Regardless of the view, one's choice of weight templates (if any) plays an important strategic role in *directing* the program's reasoning, in contrast to the use of the set of support strategy for *restricting* reasoning.

By assigning low weights (high priorities) to various functions and combinations of functions, the researcher causes an automated reasoning program to prefer for the focus of attention conclusions containing corresponding subterms. At the other end of the spectrum, the assignment of high weights (low priorities) causes the program to delay focusing on conclusions containing corresponding subterms. If one has no intuition or knowledge to be reflected with included weight templates, then the choice (for the program) of where next to focus attention can be based on symbol count. In that case, the program directs its search by selecting for the focus of attention the shortest clause that has not yet been used. If the computed weight of a deduced conclusion exceeds the assigned max_weight (limit on complexity of retained information), then the conclusion is immediately discarded.

3.4. Level Saturation and the Ratio Strategy

In contrast to the use of weighting is the use of level saturation. For this choice of directing the reasoning, one uses the command ''set(sos_queue)'', which causes the order of the selection (for the focus of attention) of retained conclusions to be based strictly on the order in which they are retained, first come first serve. In the vast majority of cases, we prefer the use of weighting (complexity preference) over that of level saturation.

However, because experimentation has showed the need for using clauses with heavy weight that are found rather early in OTTER's attack—clauses whose weight delays for a long time or forever their being chosen as the focus of attention—we frequently prefer the program's reasoning be directed by a combination of weighting and of level saturation. For that preference, we use the *ratio strategy* formulated by McCune. When one uses the ratio strategy, one assigns a value to its parameter (pick_given_ratio) for determining how much the program is to be directed by weighting and how much by level saturation. For example, by assigning pick_given_ratio to 3, OTTER is instructed to focus on three conclusions (from among those retained) based on their weight or complexity, one by first come first serve, then three, then one, and the like. Though not basing our action on a thorough analysis of which value is most effective for assigning to pick_given_ratio parameter, we note that we frequently use the value 3. If an examination of an output file suggests that too much of the program's attention is focused on first come first serve clauses, then one increases pick_given_ratio, perhaps to 6. If too little time is spent on such clauses, then one decreases pick_given_ratio to 2 or even to 1.

3.5. Weight Lists

To complete our introduction to the features of OTTER in the context of weighting, we mention three lists of interest: weight_list(pick_and_purge), weight_list(pick_given), and weight_list(purge_gen). Any weight templates included in the first list are used by the program both to direct its reasoning regarding where next to focus its attention and to affect its retention of drawn conclusions by measuring their respective desirability against the maximum (max_weight) allowed complexity. When a drawn conclusion fails to contain any subterms that match any of the included weight templates, its complexity is measured strictly in terms of symbol count. When the complexity of a drawn conclusion strictly exceeds the chosen max_weight, the conclusion is immediately purged. Weight templates included in the second list, weight_list(pick_given), are used only to direct the program's reasoning and have no effect on conclusion purging. Weight templates included in the third list, weight_list(purge_gen), are used only for conclusion purging and have no effect on directing the reasoning.

A quick review of the preceding observations correctly suggests that the first list combines the functions of the second and third. The choice of which of the three lists to use depends on whether the researcher has advice to give regarding which terms merit the assignment of high priority (small weight) for influencing OTTER on where next to focus its attention, advice to give regarding which terms merit the assignment of such low priority that their presence strongly suggests purging the corresponding drawn conclusions, and advice to give regarding the separation of included hints.

3.6. Examples of the Use of Weighting

In view of the fact that our use of the resonance strategy with OTTER is via weighting, a few remarks and a few examples of the typical use of weighting are in order. First, we note that one could add the resonance strategy to a program that does not offer weighting (in its fullest). Indeed, this strategy is based on a program's ability to treat the researcher's chosen formulas and equations to be used as resonators (see the beginning of Section 2 for the definition) to direct the program's reasoning. What is required is the ability to compare the shape of each retained conclusion with all resonators (ignoring the variables of the conclusion and of the resonator being compared) and, if any match is found, assign to the conclusion the corresponding priority given by the researcher. We were fortunate in that McCune's implementation of Overbeek's weighting strategy is such that variables in a template are treated as indistinguishable. He could, instead, have chosen (as was the case in a prior reasoning program we used) to have distinct variables treated as distinct variables, for example, requiring that the variable *x* be treated only as *x*.

For a first example of the typical use of weighting, let the function *inv* denote inverse (as in a group). If the researcher conjectures that any term in *inv* merits strong consideration, the following template can be used.

```
weight(inv(*0),1).
```

In the presence of this template, OTTER will compute the weight of any term in *inv* by multiplying the weight of its argument by 0 and adding 1. If the weight of the constant *a* is 1, then the assigned weight of *inv*(*inv*(*a*)) will be 1. If the researcher wishes terms in *inv* to be penalized, then two choices exist: the additive value can be changed from 1 to say 5, or the multiplier can be changed from 0 to say 2, or both.

For a second example, let us assume that the researcher conjectures that terms free of variables are to be preferred and, as a consequence, wishes to penalize terms containing variables. The following template (or one like it) serves nicely.

```
weight(x,3).
```

Unless some interference occurs, OTTER will add 3 to the weight of a term for each variable present in the term.

For a third example, let us assume that the researcher wishes the program to prefer for the focus of attention formulas whose ''tail'' is short, where the tail of a formula (for this discussion) is the second argument of the leftmost function *i* for implication. The following weight template is the one we use.

```
weight(P(i(*1,*2)),1).
```

In the presence of this template, OTTER assigns a weight to each new conclusion by multiplying the weight of its first argument by 1, multiplying the weight of its second argument by 2, and adding 1 to the sum of the resulting products. McCune used this template in his application of his *tail strategy* to problems in equivalential calculus, where the function *i* is replaced by the function *e* for equivalence. When we used the *recursive tail strategy* [Wos93b] in our study of two-valued sentential (or propositional) calculus, we used the following closely related template.

```
weight(i(*1,*2),1).
```

In the presence of this last template, the program gives (recursive) preference to formulas that have a short tail *regardless* of how deep in the formula the short tail is.

3.7. Additional Commands and Options

To extend our brief treatment of some of the OTTER options, we focus on the following commands found in the input file given in Section 1.5.

```
assign(max_proofs, 0).
% assign(max_distinct_vars, 4).
% clear(for_sub).
clear(back_sub).
```

The first of the given four commands assigns to the program the maximum number of proofs to be sought. When its value is set to 0, then no limit is placed on the number of proofs. Were it instead set, say, to 3, then the completion by OTTER of a third proof would terminate the run. Of course, before the desired number of completed proofs has been reached, a run may be terminated by a maximum placed on CPU time to be used or a maximum placed on memory to be used. No test is made to see whether more than one proof proves the same theorem.

Of the four commands under discussion, the second (commented out) is pertinent to a sometimes used but unnamed restriction strategy. If one removes the comment (%) and uses the command as given, OTTER is restricted in its retention of deduced conclusions by requiring that each contain no more than four distinct variables. This restriction strategy is of particular use when experiments suggest that the space of conclusions that can be drawn is large, for example, when one conjectures that long formulas must be used in turn requiring the assignment of a high max_weight.

To understand the use of the final two (of the four given) commands, we need a definition. The clause *A subsumes* the clause *B* if and only if a uniform replacement of terms for variables in *A* yields a (not necessarily proper) subclause of *B*. Forward subsumption, which is abbreviated for_sub, is the procedure OTTER employs to purge a newly generated conclusion when a subsuming clause already exists in the database of retained information. Without the use of forward subsumption, the researcher or program will almost always drown in redundant conclusions. Back subsumption is the procedure used to purge an existing conclusion when a newly deduced conclusion captures it as a corollary. When seeking proofs shorter than one has in hand, OTTER offers the use of *ancestor subsumption*. By definition, the clause *A ancestor-subsumes* the clause *B* if and only if (1) *A* properly subsumes *B* or (2) *A* and *B* are alphabetic variants and the derivation length of *A* is less than or equal to that of *B*. The question of whether the conclusion *A* subsumes, properly subsumes, or ancestor subsumes the conclusion *B* focuses on a *relation* that may or may not hold; in contrast, forward and back subsumption are *procedures* used to possibly purge conclusions.

3.8. Paramodulation, Equality, and Inference Rules

As for inference rules for drawing conclusions, OTTER offers quite a number, including binary resolution, hyperresolution, UR-resolution, and paramodulation [Wos87,Wos92], the latter generalizing the usual notion of equality substitution. By using paramodulation coupled with demodulation, OTTER offers the researcher the use of a Knuth-Bendix approach to proving theorems; see Sections 6 and 7. For a complex example of paramodulation—one that shows why a person might not enjoy applying this rule by hand—we consider the inference rule applied to both the equation $x + (-x) = 0$ and the equation $y + (-y + z) = z$; the application yields, in a single step, the conclusion $y + 0 = -(-y)$. In clause form, *from*

```
EQUAL(sum(x,minus(x)),0)
```

into

```
EQUAL(sum(y,sum(minus(y),z)),z)
```

the clause

```
EQUAL(sum(y,0),minus(minus(y)))
```

is obtained by paramodulation.

To see that this last clause is in fact a logical consequence of its two parents, one unifies the argument *sum*(*x*,*minus*(*x*)) with the term *sum*(*minus*(*y*),*z*), applies the corresponding substitution to both the *from* and *into clauses*, and then makes the appropriate term replacement justified by the typical use

of equality. The substitution found by the attempt to unify the given argument and given term requires substituting *minus*(*y*) for *x* and *minus*(*minus*(*y*)) for *z*. To prepare for the (standard) use of equality in this third example—and here we encounter a key feature of paramodulation—a nontrivial substitution for variables in both the *from* and the *into clauses* is required, which illustrates how paramodulation generalizes the usual notion of equality substitution. In contrast, in the standard use of equality substitution, one does *not* apply a nontrivial replacement for variables in both the *from* and the *into* statements.

In addition to using the set of support strategy, which is a general strategy designed to restrict the application of any inference rule, an automated reasoning program (such as OTTER) can also use a number of other strategies that are specific in that they are designed to restrict the application of paramodulation. First, the program can be restricted from paramodulating *from* or *into* a variable. This restriction prevents a myriad of conclusions from being drawn. Such a restriction strategy is, in most cases, mandatory. After all, paramodulating *from* or *into* a variable will always yield a conclusion since the corresponding unification can never fail. Second, the program can be required to paramodulate only from a specified side of each equality. In some cases, all applications of paramodulation are required to be from the left side only; in others, all are required to be from the right only.

3.9. Sources for Additional Information

The preceding material provides the merest hint of the versatility, usefulness, and power offered by McCune's program OTTER. In the book [Wos92], one gains a far greater appreciation for this program. The book also includes a tutorial (in Chapter 16) for using OTTER, as well as a diskette containing the appropriate manuals, test problems, source code, and load modules (for personal computers). We have found OTTER to be an excellent assistant for research. For but one example, as may be clear at this point, OTTER's treatment of variables in weight templates made it trivial to experiment with the resonance strategy to determine its potential. Let us now turn to some of those experiments, specifically, to those experiments motivated by the writing of this article.

4. An Iterative Use of the Resonance Strategy

In Sections 1.1 and 1.2, we discussed Scott's challenge and our response culminating in the formulation of the resonance strategy. In particular, we noted that, without the resonance strategy, 37 of 68 target theorems were proved in a single run with OTTER and, with the use of the strategy, all 68 were proved in a single run on the first attempt. In that success, which we called Experiment 0 in Section 1.2, all 68 theorems were used as resonators in the weight_list(pick_and_purge), each assigned a weight of 2. In Section 1.5, we supplied an appropriate input file for the researcher who wishes to duplicate our experiments both before and after the formulation of the resonance strategy.

Viewing the cited use of the resonance strategy as a limiting case, we were (during the writing of this article) curious about the results of so-called intermediate points. Specifically, we decided to experiment with an iterative use of the strategy. The plan was to start (with Experiment 1) by repeating the experiment that produced the 37 proofs, use as resonators in Experiment 2 the union of the proof steps of the proofs obtained in Experiment 1, use as resonators in Experiment 3 the union of the proof steps of the proofs obtained in Experiment 2, and continue in this manner. In the attempt to somewhat isolate the effects of using the strategy, the plan also called for *no* other changes to be made in the parameters. We thought it highly likely that a comparison of two proofs of the same theorem obtained in succeeding runs would show that OTTER was *not* proof checking: Two different proofs would be obtained, in many cases. By executing the cited plan, we were in part gathering evidence of the value of using as resonators the steps of the proof of one theorem in the attempt to prove a related theorem. We were also simulating (in a not totally satisfactory manner) the use of the *dynamic resonance strategy*; see Section 2.

In Experiment 1, the following 37 theses were proved.

04 05 06 09 10 11 12 13 15 16 17 18 19 20 21 22 23 24
30 36 37 38 39 40 41 42 48 49 50 62 63 64 65 66 67 68 71

When we took the union of the steps from the 37 proofs, ignoring the input (hypotheses), we obtained 83 resonators for Experiment 2. Experiment 2 yielded proofs of the following 52 theses.

04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 30 36 37 38 39 40 41 42 43 45 46 47 48 49
50 51 52 53 54 56 62 63 64 65 66 67 68 69 70 71

Also proved in the experiment was the axiom system of Hilbert, which consists of six theses, 3, 18, 21, 22, 30, and 54 (of which 30 is dependent on the other five members). (For computing the length of a proof, we ignore the axioms and count just the deduced steps; condensed detachment is the inference rule that is used in this area of logic.) As evidence that OTTER is not simply reproducing the proofs found in Experiment 1 (as in proof checking), note that, of the 37 theses proved in common, in Experiment 2 the proof length did not change in eleven of them, the proof length increased in four of them, and the proof length decreased in twenty-two of them. (Where a thesis was proved more than once, for comparison we took the shortest proof in each case.) The theses in which the proof length increased in Experiment 2 when compared with Experiment 1 are 11, 12, 13, and 63; in each case, the proof length increased by 1. Those in which the proof length decreased are theses 10, 15, 17 through 24, 30, 36 through 42, 48 through 50, and 62. Among the highlights, the length of the proof of thesis 10 was decreased from 13 to 4, of thesis 15 from 30 to 17, and of thesis 50 from 41 to 31.

For Experiment 3, we took the union of the (deduced) proof steps of the 52 proofs obtained in Experiment 2 to use as resonators. The union consists of 90 formulas, of which 14 do *not* match in functional shape any of the 83 resonators used in Experiment 2. Of the remaining 76 formulas, 68 are identical (at the entire formula level) to one of the 83 resonators, and 8 match at the resonator level but not at the entire formula level. The use of the 90 resonators (from Experiment 2) in Experiment 3 yielded 58 of the sought-after 68 proofs, the following theses.

04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 30 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 56 57 59 61 62 63 64 65 66 67
68 69 70 71

When the proofs that are obtained in both Experiment 2 and Experiment 3 are compared by length, one finds that Experiment 3 finds 15 of shorter length, 19 of longer length, and 18 of unchanged length.

In Experiment 4, we took the union of the (deduced) proof steps of the 58 proofs completed with Experiment 3, obtaining 99 formulas to use as resonators. Of the 99, 85 are identical to one of the 90 resonators used in Experiment 3, 8 differ (at the entire formula level) but match as resonators, and 6 do not even match at the resonator (functional shape) level. Using the 99 resonators, Experiment 4 yielded proofs of all but thesis 35. We were in no way surprised, for earlier experiments had produced abundant evidence that thesis 35 resists proof and resists it well, if the resonance strategy is not used. Experiment 4 also yielded a complete proof of an alternative axiom system (of Lukasiewicz) for two-valued sentential calculus, that consisting of theses 19, 37, and 59.

In Experiment 5, we took the union of the (deduced) proof steps of the 67 proofs completed with Experiment 4, obtaining 127 formulas to use as resonators. Of the 127, 97 are identical to one of the 99 resonators used in Experiment 3, 25 differ (at the entire formula level) from each of the 99 but match as resonators, and 5 do not even match at the resonator (functional shape) level. With the 127 resonators, Experiment 5 proved thesis 35, and the iterative approach yielded the desired 68 proofs. Also proved with that experiment was yet another axiom system for the calculus, namely, that consisting of theses 19, 37, and 60, discovered in our earlier research [Wos91b].

Valuable information concerning the use of the resonance strategy can be extracted from a comparison of data produced by Experiment 5, in which 127 resonators are used, with Experiment 0, in which the 68 theses are used as resonators. First, one finds that the proof of thesis 35 obtained with Experiment 5 required approximately 227 CPU-seconds on a SPARCstation 2, completing with the retention of a clause numbered 18,774. In sharp contrast, the use of the 68 theses as resonators (see Section 1.2) yielded a proof of thesis 35 in approximately 36 CPU-seconds with the retention of a clause numbered 4978. Second, if one attempts to explain the marked difference in the two experiments that each prove thesis 35, one finds in their respective proofs a most interesting property. In particular, one finds that the last two steps of the proof completed in 36 CPU-seconds have the same shape—both matching at the resonator level thesis 35—but such is not the case for the proof completed in 227 CPU-seconds. The two steps under discussion are the following (including the corresponding

history for deducing each). When OTTER is instructed to use order_history, [hyper,1,*j*,*k*] means that the inference rule *hyperresolution* is used with clause (1) to capture condensed detachment, that the clause numbered *j* is the major premiss unified with the first literal of (1), and that the clause numbered *k* is the minor premiss unified with the second literal of (1).

2595 [hyper,1,1471,1092] P(i(i(x,i(y,z)),i(i(y,x),i(y,z)))).
4978 [hyper,1,1192,2595] P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).

As the following observations show, this second discovery (of the two cited proof steps) contains the needed clue for explaining why the experiment in which the 68 theses are used as resonators is far more successful at proving thesis 35. Since thesis 35 is among the 68 theses used as resonators, any formula that matches it in shape (ignoring variables) is given preferential treatment for being chosen as the focus of attention to drive the program's reasoning. Each of the resonators was assigned a weight of 2, and thesis 35 and formulas that match it in shape have a weight of 14 if the weight is computed based on symbol count (including the predicate symbol). Therefore, the first of the cited formulas, clause (2595), is quickly chosen as the focus of attention because of being given a weight of 2. This situation contrasts sharply with Experiment 5 where, as the evidence strongly suggests, the weight of 14 delays the choice of clause (2595) as the focus of attention. By quickly choosing clause (2595) to drive OTTER's reasoning, as the ancestor history shows, clause (4978) is deduced, and that clause is in fact thesis 35. As a side effect of the delay in choosing clause (2595) to drive the reasoning, Experiment 5 completes a 56-step proof, where the more successful experiment (Experiment 0) completes a 42-step proof.

Showing how complicated is both the activity of proving deep theorems and the analysis of which parameters tell the relevant story, we note that the level of the 56-step proof is 18, and the level of the 42-step proof is 20. The level of an input statement (hypothesis) is defined to be 0, and the level of a deduced conclusion is one greater than the maximum of the levels of its immediate parents or ancestors. For example, let *A*0, *B*0, and *C*0 be (input) axioms and hence of level 0; of level 1, let *A*1 be deduced from *A*0 and *B*0, and let *B*1 be deduced from *B*0 and *C*0; and of level 2, let *A*2 be deduced from *A*1 and *B*1. Where *A*2 is a theorem of interest, by definition, its *proof level* is 2, and its *proof length* is 3; length in this report does *not* include (input) hypotheses.

Summarizing, one sees that the inclusion of thesis 35 as a resonator did cause OTTER to focus on a formula similar in shape, using that formula to markedly aid its attempt to complete a proof of thesis 35. When thesis 35 is *not* included as a resonator (as is the case in Experiment 5), the program follows a sharply different path to completing the desired proof. Indeed, a glance at the following two clauses, which are the last two steps of the proof produced by Experiment 5, shows how different the path is.

15795 [hyper,1,10712,7025] P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u))))).
18774 [hyper,1,15795,1912] P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).

Obviously, clause 15795, which is used to deduce thesis 35, does *not* match in shape thesis 35 even at the resonator level.

- We thus have an illustration of a strategy for increasing the effectiveness of the resonance strategy: *Include among the resonators the correspondent(s) of the theorem(s) that are to be proved.* In addition, effectiveness sometimes is increased by also including as resonators the hypotheses of the study.

We conducted one final experiment, Experiment 6. For resonators, we took the 125 distinct proof steps from the 68 proofs produced by Experiment 5. Since Experiment 5 used 127 resonators taken from the union of the proof steps from Experiment 4, and since Experiment 4 did not yield a proof of thesis 35 but Experiment 5 did, Experiment 5 clearly found some shorter proofs. Of the 125, 120 match identically one of the 127 resonators used in Experiment 5, 4 match at the resonator level but not at the entire formula level, and 1 matches at neither the entire formula level nor the resonator level. We were certain that Experiment 6 would yield complete proofs of various axiom systems for two-valued sentential calculus, those given in the input file of Section 1.5. The key is the presence among

the resonators of that which captures at the functional or skeletal shape thesis 35. Indeed, when OTTER deduces thesis 35, its presence as a resonator (with low weight) predictably causes the program to quickly choose thesis 35 as the focus of attention, which in turn quickly leads to completing proofs for axiom systems in which thesis 35 is a member. If one takes the union of the proof steps of the proofs produced with Experiment 6, one obtains 126 distinct steps, of which 2 fail to be identical to any of the 125 resonators that are used; however, each of the two steps in question does match in shape a resonator.

There remained the question of how different would the proofs be produced by Experiment 6 when compared with the proofs produced by Experiment 0 in which the resonators are just those corresponding (in shape) to the 68 theses under study. Table 1 compares the results of the two experiments.

Table 1: Comparison of Proofs

	Ex. 6	Ex. 0
Thesis 35		
Proof length[a]	45	42
Level	15	20
Hilbert system		
Time	64 CPU-sec	27 CPU-sec
Length	47	45
Level	14	19
Alt. Lukasiewicz system		
Time	60 CPU-sec	35 CPU-sec
Length	41	41
Level	14	19
Wos system		
Time	105 CPU-sec	76 CPU-sec
Length	43	43
Level	14	20
Church system		
Time	109 CPU-sec	78 CPU-sec
Length	46	42
Level	15	20
Frege system		
Time	78 CPU-sec	109 CPU-sec
Length	54	47
Level	15	20

[a] Both proofs complete with the same last two steps, those matching in shape thesis 35.

5. Robbins Algebra

In this section, we briefly illustrate the value of using the resonance strategy for seeking shorter proofs where the focus is on Robbins algebra. This particular study broadens the scope of using the resonance strategy, for the treatment of Robbins algebra is purely in terms of equality—condensed detachment and hyperresolution play no role and are replaced by the inference rule paramodulation; see

the example of paramodulation in action given shortly, the example discussed in more detail in Section 3.8. Part of the interest in Robbins algebra rests with the following question, still open: Is every Robbins algebra a Boolean algebra? This question was studied in some depth by Alfred Tarski and by his students, as we learned in a phone conversation with him. It is known that every *finite* Robbins algebra is Boolean.

Paramodulation [Wos87,Wos92] is the only inference rule in use. For an example of the latitude permitted in the use of paramodulation—an example showing how this inference rule generalizes the usual notion of equality substitution—we consider the following three clauses and apply paramodulation *from* the first *into* the second to yield the third; from the viewpoint of mathematics, paramodulation applied to both the equation $x + (-x) = 0$ and the equation $y + (-y + z) = z$ yields in a single step the conclusion $y + 0 = -(-y)$.

```
EQUAL(sum(x,minus(x)),0).
EQUAL(sum(y,sum(minus(y),z)),z).
EQUAL(sum(y,0),minus(minus(y))).
```

Where ''+'' can be interpreted as addition (or, more familiarly in the context of Boolean algebra, as union) and the function *n* as negation or complement, the following three axioms (expressed in yet another form acceptable to OTTER) characterize Robbins algebra.

```
EQ(+(x,y),+(y,x)).
EQ(+(+(x,y),z),+(x,+(y,z))).
EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).    % Robbins axiom
```

The first two axioms respectively express the properties of commutativity and associativity of union. The third axiom expresses an identity in terms of union and complement. All three properties hold in Boolean algebra. Regarding the cited open question, no finite counterexample (or model) can be found, for there exists a rather straightforward proof asserting that finite Robbins algebras are Boolean. One therefore has two choices to settle the open question: Find an infinite model that is a Robbins algebra and that violates one of the known properties of Boolean algebra, or prove that Robbins implies Boolean.

We learned of the open question in the early 1980s from a colleague, Steve Winker. He had been attempting, without gaining any valuable information, to use one of the Argonne automated reasoning programs to obtain a proof (rather than a model to serve as a counterexample). We suggested that he use what might be termed *binary chop*: Select some property of Boolean algebra often associated with an axiom set, adjoin the chosen property to the Robbins axioms, and then see whether the remaining properties of a Boolean algebra could be proved. When asked for a good candidate, we suggested the addition of the union of x and $x = x$. Winker succeeded, proving that the addition of this axiom to those of a Robbins algebra guarantees that the resulting algebra is Boolean. He continued to follow our suggestion of binary chop and obtained numerous intriguing results, some with the program by itself, some with the team of him and the program, and some with his insight alone. For example, the addition to the Robbins axioms of a 0 yields Boolean; the addition of a 1 yields Boolean; the addition of an axiom that asserts the existence of two elements c and d such that the union of c and $d = d$ suffices; for the Winker successes, see [Winker90,Winker92].

The last of the cited results is most fascinating, for it does indeed appear to be a weak assumption. Before Winker proved the corresponding theorem, he proved that the addition of an axiom asserting the existence of an element c with the union of c and $c = c$ suffices. This theorem is the focus of this section, in the context of finding shorter proofs by relying on the resonance strategy.

With the following input file, whose weight templates (which will be seen to serve nicely as resonators) could be commented out by placing % in column 1, one can have OTTER prove that the adjunction of $+(c,c) = c$ to the Robbins axioms is sufficient to prove Huntington's axiom. Huntington's axiom together with commutativity and associativity of + suffice to characterize Boolean algebra. Without using the resonance strategy, OTTER produces a 41-step proof (Proof 1 given shortly) from which one can extract the positive equalities, including those arising from back demodulation, to use as resonators, those resonators found in the weight_list(pick_and_purge) in the input file given shortly. When these are used as resonators, OTTER then obtains a 31-step proof (Proof 2 given shortly). In

other words, again one has an example of an action that, from a casual perspective, is simply in the spirit of proof checking: one expects the program to reproduce the proof just found—but it fails to do so. Instead, a substantially shorter proof is found, with the only change in the input being the use of the resonance strategy, keying on the steps of a known proof, Proof 1. Note that the command set(knuth_bendix) in the following input file instructs OTTER to attack the theorem under study with an approach that emphasizes the roles of paramodulation and demodulation, in a manner familiar to researchers who focus on complete sets of reductions.

Input File for Studying Robbins Algebra

```
set(knuth_bendix).
set(index_for_back_demod).
set(dynamic_demod_lex_dep).
set(lex_rpo).
set(process_input).
set(display_terms).
clear(print_kept).
clear(print_new_demod).
clear(print_back_demod).
assign(report, 60).
assign(max_weight, 20).
assign(max_mem, 8000).
lex([A, B, C, D, E, F, g(x), n(x), +(x,x)]).
lrpo_lr_status([+(x,x)]).

 weight_list(pick_and_purge).
%  following are steps from a proof of +(c,c)=c, the union of c and c = c.
weight(EQ(+(C,+(C,x)),+(C,x)), 2).
weight(EQ(+(C,+(x,C)),+(C,x)), 2).
weight(EQ(+(C,+(x,+(y,C))),+(C,+(x,y))), 2).
weight(EQ(+(C,+(x,+(C,y))),+(C,+(x,y))), 2).
weight(EQ(n(+(n(+(C,x)),n(+(C,n(+(x,C))))),C), 2).
weight(EQ(n(+(n(+(C,x)),n(+(C,n(+(C,x))))),C), 2).
weight(EQ(n(+(n(C),n(+(C,n(C))))),C), 2).
weight(EQ(n(+(n(+(x,y)),n(+(y,n(x))))),y), 2).
weight(EQ(n(+(n(+(x,y)),n(+(n(y),x)))),x), 2).
weight(EQ(n(+(C,n(+(n(C),+(C,n(C))))),n(C)), 2).
weight(EQ(+(C,+(x,y)),+(C,+(y,x))), 2).
weight(EQ(n(+(n(+(C,+(x,y))),n(+(C,n(+(y,x))))),C), 2).
weight(EQ(n(+(n(+(C,x)),n(+(n(C),+(x,C))))),+(x,C)), 2).
weight(EQ(n(+(n(+(x,+(y,z))),n(+(z,n(+(x,y))))),z), 2).
weight(EQ(n(+(n(+(x,y)),n(+(n(x),y)))),y), 2).
weight(EQ(n(+(n(+(x,n(y))),n(+(y,x)))),x), 2).
weight(EQ(n(+(n(+(n(x),y)),n(+(y,x)))),y), 2).
weight(EQ(n(+(n(+(x,+(y,z))),n(+(n(+(x,y)),z)))),z), 2).
weight(EQ(n(+(n(+(x,C)),n(+(C,n(+(C,x))))),C), 2).
weight(EQ(n(+(C,n(+(C,+(n(+(C,x)),n(+(x,C)))))),n(+(x,C))), 2).
weight(EQ(n(+(n(C),n(+(C,+(n(C),n(+(C,n(C))))))),C), 2).
weight(EQ(n(+(n(+(x,+(y,z))),n(+(y,n(+(x,z))))),y), 2).
weight(EQ(n(+(n(C),n(+(n(C),+(C,n(C))))),C), 2).
weight(EQ(n(+(n(C),+(C,n(C)))),n(+(C,n(C)))), 2).
weight(EQ(n(+(C,n(+(C,n(C))))),n(C)), 2).
weight(EQ(n(+(n(+(x,+(y,z))),n(+(n(+(x,z)),y)))),y), 2).
weight(EQ(n(+(x,C)),n(+(C,x))), 2).
weight(EQ(n(+(x,+(y,C))),n(+(C,+(x,y)))), 2).
```

```
weight(EQ(n(+(n(+(C,x)),n(+(C,+(n(C),x))))),+(x,C)), 2).
weight(EQ(n(+(n(+(C,x)),n(+(n(C),+(x,n(+(C,n(C))))))),x), 2).
weight(EQ(+(C,n(+(C,n(C)))),C), 2).
weight(EQ(+(C,+(x,n(+(C,n(C))))),+(C,x)), 2).
weight(EQ(+(x,n(+(C,n(C)))),x), 2).
weight(EQ(+(n(+(C,n(C))),x),x), 2).
weight(EQ(n(+(n(x),n(+(C,+(x,n(C)))))),x), 2).
weight(EQ(n(+(n(x),n(n(x)))),n(+(C,n(C)))), 2).
weight(EQ(n(+(x,n(x))),n(+(C,n(C)))), 2).
weight(EQ(n(n(+(x,n(n(x))))),n(n(x))), 2).
weight(EQ(n(n(x)),x), 2).
weight(EQ(+(n(+(y,x)),n(+(n(y),x))),n(x)), 2).
 end_of_list.

list(usable).
EQ(x,x).
EQ(+(x,y),+(y,x)).
EQ(+(+(x,y),z),+(x,+(y,z))).
end_of_list.

list(sos).
EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).     % Robbins axiom
EQ(+(C,C),C).   % hypothesis
-EQ(+(n(+(A,n(B))),n(+(n(A),n(B)))),B).  % denial of Huntington axiom
end_of_list.

% list(demodulators).
% (+(x,y) = +(y,x)).
% (+(+(x,y),z) = +(x,+(y,z))).
% (n(+(n(+(x,y)),n(+(x,n(y))))) = x).     % Robbins axiom
% (+(C,C) = C).   % hypothesis
% end_of_list.
```

Proof 1 in Robbins Algebra

```
----> UNIT CONFLICT at  63.07 sec ----> 909 [binary,908,1] $F.

Length of proof is 41.

---------- PROOF ----------

1 [] EQ(x,x).
3,2 [] EQ(+(x,y),+(y,x)).
5,4 [] EQ(+(+(x,y),z),+(x,+(y,z))).
6 [] EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).
9,8 [] EQ(+(C,C),C).
10 [] -(EQ(+(n(+(A,n(B))),n(+(n(A),n(B)))),B)).
----------------------------------------
12,11 [para_from,8,4] EQ(+(C,+(C,x)),+(C,x)).
14,13 [para_into,11,2] EQ(+(C,+(x,C)),+(C,x)).
15 [para_into,13,4] EQ(+(C,+(x,+(y,C))),+(C,+(x,y))).
18,17 [para_from,13,4,demod,5,5] EQ(+(C,+(x,+(C,y))),+(C,+(x,y))).
19 [para_into,6,13] EQ(n(+(n(+(C,x)),n(+(C,n(+(x,C)))))),C).
21 [para_into,6,11] EQ(n(+(n(+(C,x)),n(+(C,n(+(C,x)))))),C).
23 [para_into,6,8] EQ(n(+(n(C),n(+(C,n(C))))),C).
```

27 [para_into,6,2] EQ(n(+(n(+(x,y)),n(+(y,n(x))))),y).
31 [para_into,6,2] EQ(n(+(n(+(x,y)),n(+(n(y),x)))),x).
36,35 [para_from,23,6,demod,3] EQ(n(+(C,n(+(n(C),+(C,n(C)))))),n(C)).
43 [para_into,15,2,demod,5,18] EQ(+(C,+(x,y)),+(C,+(y,x))).
51 [para_from,43,6] EQ(n(+(n(+(C,+(x,y))),n(+(C,n(+(y,x)))))),C).
61 [para_into,27,13,demod,3] EQ(n(+(n(+(C,x)),n(+(n(C),+(x,C))))),+(x,C)).
65 [para_into,27,4] EQ(n(+(n(+(x,+(y,z))),n(+(z,n(+(x,y)))))),z).
73 [para_into,27,2] EQ(n(+(n(+(x,y)),n(+(n(x),y)))),y).
77 [para_into,27,2] EQ(n(+(n(+(x,n(y))),n(+(y,x)))),x).
99 [para_into,31,2] EQ(n(+(n(+(n(x),y)),n(+(y,x)))),y).
107 [para_into,73,4] EQ(n(+(n(+(x,+(y,z))),n(+(n(+(x,y)),z)))),z).
149 [para_into,21,2] EQ(n(+(n(+(x,C)),n(+(C,n(+(C,x)))))),C).
178,177 [para_from,149,27,demod,5,3] EQ(n(+(C,n(+(C,+(n(+(C,x)),n(+(x,C))))))),n(+(x,C))).
214,213 [para_into,51,23,demod,3,9,3] EQ(n(+(n(C),n(+(C,+(n(C),n(+(C,n(C)))))))),C).
219 [para_into,65,2] EQ(n(+(n(+(x,+(y,z))),n(+(y,n(+(x,z)))))),y).
233 [para_into,65,35,demod,3,12,3] EQ(n(+(n(C),n(+(n(C),+(C,n(C)))))),C).
244,243 [para_from,233,99,demod,3,36] EQ(n(+(n(C),+(C,n(C)))),n(+(C,n(C)))).
249 [back_demod,35,demod,244] EQ(n(+(C,n(+(C,n(C))))),n(C)).
277 [para_into,107,2] EQ(n(+(n(+(x,+(y,z))),n(+(n(+(x,z)),y)))),y).
315 [para_into,219,19,demod,3,178] EQ(n(+(x,C)),n(+(C,x))).
334,333 [para_into,315,4] EQ(n(+(x,+(y,C))),n(+(C,+(x,y)))).
337 [back_demod,61,demod,334] EQ(n(+(n(+(C,x)),n(+(C,+(n(C),x))))),+(x,C)).
418,417 [para_into,277,23,demod,3] EQ(n(+(n(+(C,x)),n(+(n(C),+(x,n(+(C,n(C))))))),x).
441 [para_into,337,249,demod,214,3] EQ(+(C,n(+(C,n(C)))),C).
453 [para_from,441,17,demod,14] EQ(+(C,+(x,n(+(C,n(C))))),+(C,x)).
482,481 [para_from,453,77,demod,3,3,418] EQ(+(x,n(+(C,n(C)))),x).
496,495 [para_into,481,2] EQ(+(n(+(C,n(C))),x),x).
510,509 [para_from,481,219,demod,3] EQ(n(+(n(x),n(+(C,+(x,n(C)))))),x).
513 [para_from,481,99,demod,496,3] EQ(n(+(n(x),n(n(x)))),n(+(C,n(C)))).
765 [para_into,513,509,demod,510] EQ(n(+(x,n(x))),n(+(C,n(C)))).
828,827 [para_from,765,99,demod,3,496] EQ(n(n(+(x,n(n(x))))),n(n(x))).
830,829 [para_from,765,99,demod,3,482,828] EQ(n(n(x)),x).
895,894 [para_into,829,73] EQ(+(n(+(y,x)),n(+(n(y),x))),n(x)).
908 [back_demod,10,demod,895,830] -(EQ(B,B)).

Clause (908) contradicts clause (1), and the proof is complete.

Proof 2 in Robbins Algebra

----> UNIT CONFLICT at 12.38 sec ----> 477 [binary,476,1] $F.

Length of proof is 31.

---------- PROOF ----------

1 [] EQ(x,x).
3,2 [] EQ(+(x,y),+(y,x)).
5,4 [] EQ(+(+(x,y),z),+(x,+(y,z))).
6 [] EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).
8 [] EQ(+(C,C),C).
10 [] -(EQ(+(n(+(A,n(B))),n(+(n(A),n(B)))),B)).
----
13 [para_into,6,2] EQ(n(+(n(+(x,y)),n(+(y,n(x))))),y).
17 [para_into,6,2] EQ(n(+(n(+(x,y)),n(+(n(y),x)))),x).
21 [para_into,13,4] EQ(n(+(n(+(x,+(y,z))),n(+(z,n(+(x,y)))))),z).

25 [para_into,13,2] EQ(n(+(n(+(x,y)),n(+(n(x),y)))),y).
29 [para_into,13,2] EQ(n(+(n(+(x,n(y))),n(+(y,x)))),x).
47 [para_into,17,2] EQ(n(+(n(+(n(x),y)),n(+(y,x)))),y).
55 [para_into,21,2] EQ(n(+(n(+(x,+(y,z))),n(+(y,n(+(x,z)))))),y).
83 [para_into,55,2,demod,5] EQ(n(+(n(+(x,+(y,z))),n(+(x,n(+(z,y)))))),x).
85 [para_into,55,2] EQ(n(+(n(+(x,+(y,z))),n(+(n(+(x,z)),y)))),y).
115 [para_from,8,47,demod,3,3] EQ(n(+(n(C),n(+(C,n(C))))),C).
122,121 [para_from,8,4] EQ(+(C,+(C,x)),+(C,x)).
123 [para_from,115,83,demod,3,3] EQ(n(+(C,n(+(n(C),+(C,n(C)))))),n(C)).
136,135 [para_from,115,85,demod,3] EQ(n(+(n(+(C,x)),n(+(n(C),+(x,n(+(C,n(C))))))),x).
172,171 [para_into,121,2] EQ(+(C,+(x,C)),+(C,x)).
196,195 [para_from,123,55,demod,122,3] EQ(n(+(n(C),n(+(n(C),+(C,n(C)))))),C).
200,199 [para_from,123,29,demod,3,196] EQ(n(+(n(C),+(C,n(C)))),n(+(C,n(C)))).
215 [back_demod,123,demod,200] EQ(n(+(C,n(+(C,n(C))))),n(C)).
218,217 [para_into,135,8] EQ(n(+(n(C),n(+(n(C),+(C,n(+(C,n(C))))))),C).
231 [para_from,171,29,demod,3,3] EQ(n(+(n(+(C,x)),n(+(n(C),+(x,C))))),+(x,C)).
237 [para_from,171,4,demod,5,5] EQ(+(C,+(x,+(C,y))),+(C,+(x,y))).
271 [para_into,231,215,demod,3,218,3] EQ(+(C,n(+(C,n(C)))),C).
293 [para_from,271,237,demod,172] EQ(+(C,+(x,n(+(C,n(C))))),+(C,x)).
328,327 [para_from,293,29,demod,3,3,136] EQ(+(x,n(+(C,n(C)))),x).
342,341 [para_into,327,2] EQ(+(n(+(C,n(C))),x),x).
354,353 [para_from,327,55,demod,3] EQ(n(+(n(x),n(+(C,+(x,n(C)))))),x).
357 [para_from,327,29,demod,342,3] EQ(n(+(n(x),n(n(x)))),n(+(C,n(C)))).
369 [para_into,357,353,demod,354] EQ(n(+(x,n(x))),n(+(C,n(C)))).
412,411 [para_from,369,47,demod,3,328] EQ(n(n(+(x,n(n(x))))),x).
416,415 [para_from,369,47,demod,3,342,412] EQ(n(n(x)),x).
467,466 [para_into,415,25] EQ(+(n(+(y,x)),n(+(n(y),x))),n(x)).
476 [back_demod,10,demod,467,416] -(EQ(B,B)).

Clause (476) contradicts clause (1), and the proof is complete.

6. Group Theory

In this section, the focus is on group theory, mainly studying the use of the resonance strategy for seeking shorter proofs when applied to theorems concerned with single axioms. Even though dependency exists among them, for the axioms for a group, we prefer the set consisting of left and right identity with the identity denoted by *e*, left and right inverse, and associativity. The axioms of right identity and right inverse, for example, can be deduced from the remaining set of axioms. We also prefer the study of group theory to be made with the use of the inference rule *paramodulation* [Wos87,Wos92], which necessitates the inclusion of a clause for reflexivity of equality. We typically use the following clauses with *f* denoting product and *g* denoting inverse.

Left identity:
(1) EQUAL(f(e,x),x).
Right identity:
(2) EQUAL(f(x,e),x).
Left inverse:
(3) EQUAL(f(g(x),x),e).
Right inverse:
(4) EQUAL(f(x,g(x)),e).
Associativity:
(5) EQUAL(f(f(x,y),z),f(x,f(y,z))).
Reflexivity:
(6) EQUAL(x,x).

We were motivated to make these studies in part by our colleague McCune. He had used OTTER in an impressively successful search for single axioms for groups and for varieties of groups

[McCune93]. A variety of groups is the set of all groups that satisfy the usual axioms for a group and also satisfy one or more additional equations. For example, if one adds the equation asserting that $xy = yx$ for all x and y, one obtains the variety of groups known as commutative, or Abelian, groups. If, instead, one adds the equation asserting that the square of every x is the identity e, one has the variety of groups known as Boolean groups, or groups of exponent 2.

For an equation to be a single axiom for a given variety, one must be able to deduce from it the usual axioms for a group (or their equivalent) as well as the additional equation(s) that define the variety, and one must also be able to prove that the proposed axiom is satisfied by *all* members of the variety. Especially regarding the latter property, one rejects a candidate single axiom when one can prove properties that are too strong. For example, for the variety of groups in which one adds the equation asserting that the square of x commutes with the square of y, being able to prove commutativity from a proposed single axiom shows that the candidate must be rejected, for there exist groups in the variety under discussion that are not commutative, which in turn implies that the proposed axiom is not true for all members of the variety. McCune's intriguing results caused us to join him in the use of his program to extend the study to additional varieties; we succeeded.

When by e-mail the preceding paragraph was sent to Ken Kunen, he replied [Kunen93] with the following important distinctions, giving the *computer science* perspective in comparison with the mathematical perspective. In particular, Kunen remarked that although the paragraph is logically correct, it fails to accurately capture what is done by him and by us when studying such problems. Next, Kunen noted that satisfaction by all members of the variety is equivalent to deducibility of the proposed single axiom from the usual axioms for group theory together with the equation(s) that define the variety. In contrast to the spirit of the preceding paragraph, our approach is ''generate and test'', generating candidates that obviously do satisfy all of the members of the variety. (For example, McCune had OTTER take the axioms for a group and the defining equation and deduce a large set of conclusions from which he selected those he thought promising.) We accept a candidate single axiom if we can prove the needed properties from it, and we reject a candidate if we can find an appropriate model to serve as a counterexample. Models and counterexamples are found with Slaney's program FINDER [Slaney92], with Prolog programs written for that objective, and with the researcher's intuition. Kunen summarized by noting that, since we consider only those candidates that are satisfied by all members of the variety under study, we never reject a candidate because of being too strong but, instead, reject it because of being too weak.

In addition to the given motivation, we were quite curious about the usefulness of the resonance strategy for seeking shorter proofs for theorems in group theory. In particular—especially since the resonance strategy was developed exclusively in the context of the study of logic calculi, where equality (in the manner discussed here) plays no role—would the methodology apply well to problems in which equality is the *only* relationship present?

The area of study concerns the variety of groups of exponent 3, those in which the cube of every element x is the identity e. Although McCune had mainly sought single axioms in which the identity is present implicitly, in deference to our preference, he then initiated the search for single axioms in which the identity e is present explicitly. He had OTTER assist in finding good candidates and then, for each, study it to see whether the appropriate theorems could be proved. For exponent 3 groups—as the input file given shortly shows—one is asked to prove that associativity holds (for multiplication), that the element e is in fact a two-sided identity, and that the cube of every element x is e. No need exists for considering the inverse property, for the square of x is the inverse of x in the variety under discussion. Among OTTER's successes, the following equation (expressed as a clause with the function f denoting product) was proved to be a single axiom for groups of exponent 3.

EQ(f(x,f(f(f(x,y),z)),f(e,f(z,z)))),y).

In other words, where the function f denotes multiplication, OTTER proved (in less than 2 CPU-seconds) from the given single equation that the constant e is a two-sided identity ($ex = x = xe$), that multiplication is associative, and that the cube of every x is e. From the viewpoint of automated reasoning, since paramodulation is the inference rule in use, the input also contains a clause for reflexivity, $x = x$.

For our study of the variety of groups of exponent 3 and of the value of using the resonance strategy to seek shorter proofs, we conducted two experiments. In the first, we gave no guidance (with weight templates), simply asking OTTER to (in effect) reproduce McCune's success. In the second experiment, we modified the first experiment by including resonators (weight templates) to direct OTTER's search. The templates correspond to the union of the proof steps of the four proofs produced with the first experiment. The researcher who wishes to duplicate these two experiments can use the following input file, commenting out (with %) the 18 lines of weight templates for the first experiment, and leaving them (without comment) for the second.

Input File for Studying Exponent 3 Groups

```
set(knuth_bendix).
set(process_input).
set(index_for_back_demod).
% Following 8 are knuth-bendix options
% set(dynamic_demod_all).
% clear(para_from_right).
% set(back_demod).
% set(para_from).
% set(para_into).
% clear(para_into_right).
% set(dynamic_demod).
% set(order_eq).
set(lex_rpo).
lex([e,f(x,x),g(x)]).
lrpo_lr_status([f(x,x)]).
assign(pick_given_ratio, 3).
assign(max_proofs, 4).
assign(max_mem, 12000).
% set(control_memory).
set(print_level).
% set(sos_queue).
assign(max_weight, 80).
% assign(max_seconds, 3600).
assign(report, 300).
assign(demod_limit, 0).
assign(reduce_weight_limit, 1225).
clear(print_kept).
clear(print_new_demod).
clear(print_back_demod).

weight_list(pick_and_purge).
%  following is the union of proof steps for exp3.
weight(EQ(f(e,e),e), 2).
weight(EQ(f(e,f(x,f(x,f(x,e)))),e), 2).
weight(EQ(f(f(x,e),f(f(x,e),f(x,e))),e), 2).
weight(EQ(f(f(x,e),f(f(x,e),f(x,f(f(f(e,f(x,e)),y),f(y,y))))),f(x,e)), 2).
weight(EQ(f(f(x,y),e),f(x,f(y,f(z,f(z,f(z,e)))))), 2).
weight(EQ(f(f(x,y),f(e,z)),f(x,f(y,z))), 2).
weight(EQ(f(f(x,y),f(f(x,y),f(x,f(y,e)))),e), 2).
weight(EQ(f(x,e),x), 2).
weight(EQ(f(x,f(f(x,f(f(x,y),f(e,f(f(e,z),f(e,z))))),z)),y), 2).
weight(EQ(f(x,f(f(x,f(f(x,y),f(e,f(z,z)))),f(f(e,f(f(e,z),u)),f(e,f(u,u))))),y), 2).
weight(EQ(f(x,f(f(x,f(f(x,y),z)),f(f(e,z),f(e,z)))),y), 2).
weight(EQ(f(x,f(f(y,z),u)),f(f(x,y),f(f(e,z),u))), 2).
```

```
weight(EQ(f(x,f(x,f(f(x,f(y,f(z,e))),f(f(f(e,e),u),f(u,u)))),f(y,z)), 2).
weight(EQ(f(x,f(x,f(f(y,z),f(z,z)))),f(x,f(x,f(y,e)))), 2).
weight(EQ(f(x,f(x,f(x,e))),e), 2).
weight(EQ(f(x,f(x,f(x,f(y,e)))),y), 2).
weight(EQ(f(x,f(x,f(x,f(y,f(z,e))))),f(y,z)), 2).
weight(EQ(f(x,f(y,f(e,f(f(e,f(z,z)),f(e,f(z,z))))))),f(f(x,y),z)), 2).
end_of_list.

list(usable).
EQ(x,x).
end_of_list.

list(sos).
EQ(f(x,f(f(x,f(f(x,y),z)),f(e,f(z,z)))),y).
end_of_list.

list(passive).
-EQ(f(e,a),a) | $ANS(lid).
-EQ(f(a,e),a) | $ANS(rid).
-EQ(f(a,f(a,a)),e) | $ANS(exp3).
-EQ(f(f(a,b),c), f(a,f(b,c))) | $ANS(assoc).
end_of_list.
```

Both experiments prove (in order) the properties of right identity, exponent 3, left identity, and associativity. In the first experiment (without the use of the resonance strategy), the four proofs (in order) have length 9 and level 10, length 12 and level 13, length 30 and level 17, and length 31 and level 19. In the second experiment, which uses the resonance strategy, the four proofs (in order) have length 8 and level 8, length 10 and level 10, length 12 and level 12, and length 17 and level 14. We freely admit that this set of experiments was indeed satisfying, for it added to the growing evidence of the power of the resonance strategy.

7. Successfully Searching for Elegant Proofs

Although the precise definition of elegance for a proof may never be available, three relevant properties come to mind. For each of the three, we give in this section examples of using the resonance strategy to obtain a proof more elegant than that which prompted the study.

Proof length is the first of the three properties. Everything being equal, the shorter the proof, the more elegant. Of course, when specific inference rules are not in use, then the concept of the length of a proof becomes somewhat hazy; for example, symmetry of equality can be used implicitly without counting the corresponding proof step. Such potential haziness is not a problem for us in this section, for our examples each rely on the specific use of condensed detachment. More generally, with the exception of demodulation, no problem exists for OTTER, for by necessity this program uses specific inference rules. Regarding demodulation, typically its applications do not contribute to the reported length of a proof; for example, in proofs of single axioms for groups [Wos93a], some steps rely on more than 500 applications of demodulation, none of which are counted in the reported length, for the cited applications of demodulation merely produce intermediate steps not corresponding to retained clauses.

The second property concerns the structure of a proof, specifically, concerns the nature of the terms present in the deduced steps. For example, one might seek a proof in which absent are terms of the form $n(n(t))$ for any term t, where the function n denotes *negation*. Intuition and experience might reasonably suggest that such a goal may indeed be unreachable (or computationally out of reach) when axioms in which the function n is present are used in the proof. After all, one commonly sees the use of lemmas of the form **not**(**not**(x)) = x (in circuit design) and *minus*(*minus*(x)) = x (in mathematical proof). For a second example, logicians sometimes seek proofs that are *organic*, meaning that none of the steps of the proof contains a subformula that is true under all assignments of true and false.

We think of the third property of elegance of a proof as one of *compactness* of the proof. Rather than a formal definition, the following illustration suffices, for one can extract from it (if desired) the appropriate formalism. Let the theorem **T** under consideration be of the form *P* implies *Q* and *r* and *S*. For example, let *P* be the Lukasiewicz axiom system consisting of *L*1, *L*2, and *L*3, and, respectively, let *Q*, *R*, and *S* be theses 18, 39, and 49 (the Church axiom system). If by hand or by program one finds a proof of **T** such that the proof is a proof of exactly one of *Q* or *R* or *S*, then one has a proof that is *compact*. Of course, to be a proof of, say, *R* requires that the last step be *R* and that all steps be needed in the proof. Therefore, a compact proof of **T** that completes with the deduction of *R* must contain as subproofs proofs of each of *Q* and *S*.

As further evidence of the value of the resonance strategy—although sometimes assisted with other procedures—we give examples of proofs in which one or more of the three properties of elegance is present, those regarding *length*, *structure*, and *compactness*. The proofs are taken from two-valued sentential (or propositional) calculus and from many-valued sentential calculus, the latter being weaker than the former in the sense that the axioms of the former imply the axioms of the latter. Rather than a thorough treatment of our attack, we merely supply brief commentary to accompany the proofs.

7.1. Two-Valued Sentential Calculus

The following three 22-step proofs of the Church axiom system were each obtained starting with the Lukasiewicz axiom system consisting of *L*1, *L*2, and *L*3. The original Lukasiewicz proof is (in effect) of length 33. The first of the three proofs is (from the viewpoint of our research) chronologically the oldest.

Proof 1 in Two-Valued Sentential Calculus

```
-----> EMPTY CLAUSE at   1.21 sec ----> 122 [hyper,4,105,119,102]
$ANSWER(step_allBEH_Church_FL_18_35_49).

Length of proof is 22.  Level of proof is 16.

---------- PROOF ----------

1 [] -P(i(x,y))| -P(x)|P(y).
4 [] -P(i(p,i(q,p)))| -P(i(i(p,i(q,r)),i(i(p,q),i(p,r))))| -P(i(i(n(p),n(q)),i(q,p)))|
$ANSWER(step_allBEH_Church_FL_18_35_49).
10 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
11 [] P(i(i(n(x),x),x)).
12 [] P(i(x,i(n(x),y))).
----------------------------------------
81 [hyper,1,10,10] P(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
83 [hyper,1,10,11] P(i(i(x,y),i(i(n(x),x),y))).
85 [hyper,1,10,12] P(i(i(i(n(x),y),z),i(x,z))).
87 [hyper,1,12,11] P(i(n(i(i(n(x),x),x)),y)).
88 [hyper,1,81,81] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
91 [hyper,1,81,85] P(i(i(x,n(y)),i(y,i(x,z)))).
92 [hyper,1,88,83] P(i(i(x,i(n(y),y)),i(i(y,z),i(x,z)))).
93 [hyper,1,91,87] P(i(x,i(n(i(i(n(y),y),y)),z))).
94 [hyper,1,92,93] P(i(i(i(i(n(x),x),x),y),i(z,y))).
95 [hyper,1,81,94] P(i(i(x,i(n(y),y)),i(z,i(x,y)))).
98 [hyper,1,81,95] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
101 [hyper,1,92,98] P(i(i(i(i(x,y),y),z),i(i(n(y),x),z))).
102 [hyper,1,101,85] P(i(i(n(x),n(y)),i(y,x))).
104 [hyper,1,101,81] P(i(i(n(i(x,y)),i(z,y)),i(i(x,z),i(x,y)))).
105 [hyper,1,85,102] P(i(x,i(y,x))).
108 [hyper,1,10,105] P(i(i(i(x,y),z),i(y,z))).
```

110 [hyper,1,108,102] P(i(n(x),i(x,y))).
112 [hyper,1,10,110] P(i(i(i(x,y),z),i(n(x),z))).
114 [hyper,1,112,112] P(i(n(i(x,y)),i(n(x),z))).
115 [hyper,1,92,114] P(i(i(x,y),i(n(i(x,z)),y))).
117 [hyper,1,10,115] P(i(i(i(n(i(x,y)),z),u),i(i(x,z),u))).
119 [hyper,1,117,104] P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
122 [hyper,4,105,119,102] $ANSWER(step_allBEH_Church_FL_18_35_49).

Proof 2 in Two-Valued Sentential Calculus

-----> EMPTY CLAUSE at 2.06 sec ----> 139 [hyper,4,110,137,106] $ANSWER(step_allBEH_Church_FL_18_35_49).

Length of proof is 22. Level of proof is 15.

---------- PROOF ----------

1 [] -P(i(x,y))|-P(x)|P(y).
4 [] -P(i(p,i(q,p)))|-P(i(i(p,i(q,r)),i(i(p,q),i(p,r))))|-P(i(i(n(p),n(q)),i(q,p)))|$ANSWER(step_allBEH_Church_FL_18_35_49).
10 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
11 [] P(i(i(n(x),x),x)).
12 [] P(i(x,i(n(x),y))).
----
81 [hyper,1,10,10] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
83 [hyper,1,10,11] P(i(i(x,y),i(i(n(x),x),y))).
85 [hyper,1,10,12] P(i(i(i(n(x),y),z),i(x,z))).
87 [hyper,1,12,11] P(i(n(i(i(n(x),x),x)),y)).
88 [hyper,1,81,81] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
90 [hyper,1,81,83] P(i(i(x,y),i(i(n(i(y,z)),i(y,z)),i(x,z)))).
92 [hyper,1,10,87] P(i(i(x,y),i(n(i(i(n(z),z),z)),y))).
93 [hyper,1,88,90] P(i(i(x,i(n(i(y,z)),i(y,z))),i(i(u,y),i(x,i(u,z))))).
94 [hyper,1,85,92] P(i(x,i(n(i(i(n(y),y),y)),z))).
95 [hyper,1,93,94] P(i(i(x,i(n(y),y)),i(z,i(x,y)))).
96 [hyper,1,81,95] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
100 [hyper,1,93,96] P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))).
105 [hyper,1,88,100] P(i(i(x,i(n(y),z)),i(i(u,i(z,y)),i(x,i(u,y))))).
106 [hyper,1,100,12] P(i(i(n(x),n(y)),i(y,x))).
110 [hyper,1,85,106] P(i(x,i(y,x))).
115 [hyper,1,10,110] P(i(i(i(x,y),z),i(y,z))).
119 [hyper,1,115,106] P(i(n(x),i(x,y))).
122 [hyper,1,100,119] P(i(i(n(x),y),i(n(y),x))).
126 [hyper,1,122,119] P(i(n(i(x,y)),x)).
132 [hyper,1,10,126] P(i(i(x,y),i(n(i(x,z)),y))).
134 [hyper,1,105,132] P(i(i(x,i(y,i(z,u))),i(i(z,y),i(x,i(z,u))))).
137 [hyper,1,134,10] P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
139 [hyper,4,110,137,106] $ANSWER(step_allBEH_Church_FL_18_35_49).

Proof 3 in Two-Valued Sentential Calculus

-----> EMPTY CLAUSE at 1.18 sec ----> 125 [hyper,4,110,123,108] $ANSWER(step_allBEH_Church_FL_18_35_49).

Length of proof is 22. Level of proof is 17.

```
---------- PROOF ----------

1 [] -P(i(x,y))| -P(x)|P(y).
4 [] -P(i(p,i(q,p)))| -P(i(i(p,i(q,r)),i(i(p,q),i(p,r))))| -P(i(i(n(p),n(q)),i(q,p)))|
$ANSWER(step_allBEH_Church_FL_18_35_49).
10 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
11 [] P(i(i(n(x),x),x)).
12 [] P(i(x,i(n(x),y))).
----------------------------------------
81 [hyper,1,10,10] P(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
84 [hyper,1,10,12] P(i(i(i(n(x),y),z),i(x,z))).
86 [hyper,1,12,11] P(i(n(i(i(n(x),x),x)),y)).
87 [hyper,1,81,81] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
89 [hyper,1,81,10] P(i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))).
91 [hyper,1,87,89] P(i(i(x,i(i(y,z),u)),i(i(y,v),i(x,i(i(v,z),u))))).
93 [hyper,1,89,86] P(i(i(i(n(i(i(n(x),x),x)),y),z),i(i(u,y),z))).
94 [hyper,1,93,11] P(i(i(x,i(i(n(y),y),y)),i(i(n(y),y),y))).
96 [hyper,1,84,94] P(i(x,i(i(n(y),y),y))).
99 [hyper,1,91,96] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
101 [hyper,1,10,99] P(i(i(i(x,i(i(y,z),z)),u),i(i(n(z),y),u))).
103 [hyper,1,101,11] P(i(i(n(x),y),i(i(y,x),x))).
105 [hyper,1,87,103] P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))).
107 [hyper,1,91,105] P(i(i(n(x),y),i(i(z,i(u,x)),i(i(y,u),i(z,x))))).
108 [hyper,1,105,12] P(i(i(n(x),n(y)),i(y,x))).
110 [hyper,1,84,108] P(i(x,i(y,x))).
113 [hyper,1,10,110] P(i(i(i(x,y),z),i(y,z))).
115 [hyper,1,113,108] P(i(n(x),i(x,y))).
117 [hyper,1,105,115] P(i(i(n(x),y),i(n(y),x))).
119 [hyper,1,117,115] P(i(n(i(x,y)),x)).
121 [hyper,1,107,119] P(i(i(x,i(y,i(z,u))),i(i(z,y),i(x,i(z,u))))).
123 [hyper,1,121,10] P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
125 [hyper,4,110,123,108] $ANSWER(step_allBEH_Church_FL_18_35_49).
```

An examination of the three proofs yields (to us) some surprising differences. The first proof contains eight steps not present in the second, and it contains twelve steps not present in the third. The second proof contains eight steps not present in the third. Where the first and third proofs are free of any formula in which five or more distinct variables occur, the second proof contains one step in which five distinct variables occur. As for similarities, no step in any of the three proofs contains a formula of the form $n(n(t))$ for any term t, nor is there present a subterm of the form $i(x,x)$. Each proof completes with the deduction of thesis 35; therefore, each is compact. We know of no shorter proof than one of length 22, and we offer the corresponding question for research.

7.2. Many-Valued Sentential Calculus

As further evidence of the value of using the resonance strategy to search for elegant proofs, we turn to many-valued sentential calculus, which can be axiomatized with the following four formulas [Lukasiewicz63,Lukasiewicz70] (written in clause notation) in which the function i can be interpreted as ''implication'' and the function n as ''negation''.

```
P(i(x,i(y,x))).
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(i(x,y),y),i(i(y,x),x))).
P(i(i(n(x),n(y)),i(y,x))).
```

As is typical of many logic calculi, again the inference rule that is used is condensed detachment, captured by the earlier cited clause (in Sections 1.1 and 1.5, for example) for that purpose coupled with the use of hyperresolution. Many-valued sentential calculus is weaker than two-valued sentential calculus

in that the axioms of the former are deducible from the axioms of the latter. The theorem under study asserts that the following formula (in clause notation) is deducible from the given four axioms; we call the formula *MV*5.

P(i(i(i(x,y),i(y,x)),i(y,x))).

When Meredith proved that *MV*5 is dependent on the given four axioms for many-valued sentential calculus [Lukasiewicz70], it marked an important advance for that field.

From the historical perspective, the goal of proving the given theorem in a single run without substantial guidance eluded our numerous attempts with OTTER under diverse option settings for more than two and one-half years. As reported in [Wos91a], we had with an iterated approach and with many suggestions obtained a 63-step proof. Even with that proof in hand, we could go no further from February 1990 until September 1992, unless we resorted to a proof-checking mode. (For those researchers interested in a successful attack in proving the theorem under discussion *without* substantial guidance, see Section 5.2 of [Wos93b].)

We now give the input file that does this proof checking—in a manner that attempts to exclude from retention almost all other clauses—and follow it with the 63-step proof. We note that list(passive) is used only for unit conflict and for forward subsumption. (The input file and proof differ only slightly from that respectively used and obtained in February 1990; the main difference rests with the use of order_history, resulting in the ancestors being listed as nucleus, major premiss, and minor premiss.)

Input File for Proof Checking a 63-Step Proof

```
set(hyper_res).
assign(fpa_literals,8).
assign(max_weight,2).
assign(max_proofs,0).
clear(print_kept).
% clear(back_sub).
% assign(max_seconds,7200).
assign(max_mem,24000).
assign(report,300).
set(order_history).
set(sos_queue).

weight_list(purge_gen).
weight(P(i(x,i(y,i(z,y)))),2).
weight(P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))),2).
weight(P(i(x,i(i(y,z),i(i(z,u),i(y,u))))),2).
weight(P(i(i(i(x,y),z),i(y,z))),2).
weight(P(i(i(i(i(x,y),y),z),i(i(i(y,x),x),z))),2).
weight(P(i(i(i(x,y),z),i(i(n(y),n(x)),z))),2).
weight(P(i(i(i(x,i(y,x)),z),z)),2).
weight(P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))),2).
weight(P(i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))),2).
weight(P(i(i(i(i(x,y),i(i(y,z),i(x,z))),u),u)),2).
weight(P(i(n(x),i(x,y))),2).
weight(P(i(x,i(i(x,y),y))),2).
weight(P(i(i(i(i(x,i(y,x)),i(y,x)),i(i(i(x,y),y),x))),2).
weight(P(i(i(i(n(x),n(i(y,i(z,y)))),x)),2).
weight(P(i(i(i(n(x),n(i(i(y,z),i(i(z,u),i(y,u))))),x)),2).
weight(P(i(i(x,i(y,z)),i(x,i(i(z,u),i(y,u))))),2).
weight(P(i(i(x,y),i(n(y),i(x,z)))),2).
weight(P(i(i(i(x,y),z),i(n(x),z))),2).
weight(P(i(i(x,i(y,z)),i(y,i(x,z)))),2).
```

```
weight(P(i(i(x,y),i(i(i(y,x),x),y))),2).
weight(P(i(i(i(i(n(x),n(i(y,i(z,y)))),x),u),u)),2).
weight(P(i(x,i(i(y,z),i(i(x,y),z)))),2).
weight(P(i(i(i(n(x),i(y,z)),u),i(i(y,x),u))),2).
weight(P(i(n(n(x)),x)),2).
weight(P(i(i(x,y),i(i(z,x),i(z,y)))),2).
weight(P(i(i(x,i(n(y),n(i(z,i(u,z))))),i(x,y))),2).
weight(P(i(i(i(i(x,y),i(i(z,x),y)),u),i(z,u))),2).
weight(P(i(i(i(n(n(x)),y),z),i(i(x,y),z))),2).
weight(P(i(x,n(n(x)))),2).
weight(P(i(i(x,n(n(y))),i(x,y))),2).
weight(P(i(i(x,i(i(y,z),u)),i(x,i(z,u)))),2).
weight(P(i(i(x,i(n(y),n(z))),i(x,i(z,y)))),2).
weight(P(i(i(n(x),y),i(n(y),x))),2).
weight(P(i(i(n(n(x)),y),i(x,y))),2).
weight(P(i(i(i(x,y),n(z)),i(z,x))),2).
weight(P(i(i(x,y),i(n(y),n(x)))),2).
weight(P(i(i(x,i(n(n(y)),z)),i(x,i(y,z)))),2).
weight(P(i(i(x,i(y,z)),i(x,i(n(z),n(y))))),2).
weight(P(i(n(i(x,y)),n(y))),2).
weight(P(i(i(i(n(x),n(y)),z),i(i(y,x),z))),2).
weight(P(i(n(i(x,y)),n(i(i(y,z),n(x))))),2).
weight(P(i(i(i(x,y),i(z,u)),i(y,i(n(u),n(z))))),2).
weight(P(i(i(i(n(x),n(i(y,x))),n(i(y,x))),n(x))),2).
weight(P(i(i(x,i(i(n(y),n(z)),u)),i(x,i(i(z,y),u)))),2).
weight(P(i(i(i(n(x),n(y)),n(y)),i(i(x,y),n(x)))),2).
weight(P(i(i(x,n(i(y,z))),i(x,n(i(i(z,u),n(y)))))),2).
weight(P(i(i(x,i(i(n(y),n(i(z,y))),n(i(z,y)))),i(x,n(y)))),2).
weight(P(i(n(x),i(n(n(y)),n(i(y,x))))),2).
weight(P(i(i(i(i(x,y),y),n(i(x,y))),n(y))),2).
weight(P(i(n(x),i(y,n(i(y,x))))),2).
weight(P(i(i(x,i(i(i(y,z),z),n(i(y,z)))),i(x,n(z)))),2).
weight(P(i(i(i(n(x),i(y,n(i(y,x)))),z),z)),2).
weight(P(i(i(i(i(x,y),y),n(i(y,x))),n(x))),2).
weight(P(i(i(x,i(i(i(y,z),z),n(i(z,y)))),i(x,n(y)))),2).
weight(P(i(i(x,y),i(i(y,n(i(y,x))),n(x)))),2).
weight(P(i(i(i(x,n(i(x,y))),n(i(i(x,n(i(x,y))),n(y)))),n(n(y)))),2).
weight(P(i(i(i(x,n(i(x,y))),n(i(i(x,n(i(x,y))),n(y)))),y)),2).
weight(P(i(i(x,i(i(y,n(i(y,z))),n(i(y,n(i(y,z))),n(z)))),i(x,z))),2).
weight(P(i(i(i(x,n(i(x,y))),n(i(y,x))),y)),2).
weight(P(i(i(x,i(i(y,n(i(y,z))),n(i(z,y)))),i(x,z))),2).
weight(P(i(x,i(i(n(i(x,y)),n(i(y,x))),y))),2).
weight(P(i(x,i(i(i(y,x),i(x,y)),y))),2).
weight(P(i(i(i(x,y),i(y,x)),i(y,x))),2).
end_of_list.

list(axioms).
-P(i(x,y)) | -P(x) | P(y).
end_of_list.

list(sos).
P(i(x,i(y,x))).
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(i(x,y),y),i(i(y,x),x))).
P(i(i(n(x),n(y)),i(y,x))).
```

end_of_list.

list(passive).
-P(i(a,i(b,i(c,b)))) | $ANS(step_01).
-P(i(i(i(i(a,b),i(c,b)),d),i(i(c,a),d))) | $ANS(step_02).
-P(i(a,i(i(b,c),i(i(c,d),i(b,d))))) | $ANS(step_03).
-P(i(i(i(a,b),c),i(b,c))) | $ANS(step_04).
-P(i(i(i(i(a,b),b),c),i(i(i(b,a),a),c))) | $ANS(step_05).
-P(i(i(i(a,b),c),i(i(n(b),n(a)),c))) | $ANS(step_06).
-P(i(i(i(a,i(b,a)),c),c)) | $ANS(step_07).
-P(i(i(a,i(b,c)),i(i(d,b),i(a,i(d,c))))) | $ANS(step_08).
-P(i(i(a,b),i(i(i(a,c),d),i(i(b,c),d)))) | $ANS(step_09).
-P(i(i(i(i(a,b),i(i(b,c),i(a,c))),d),d)) | $ANS(step_10).
-P(i(n(a),i(a,b))) | $ANS(step_11).
-P(i(a,i(i(a,b),b))) | $ANS(step_12).
-P(i(i(i(a,i(b,a)),i(b,a)),i(i(i(a,b),b),a))) | $ANS(step_13).
-P(i(i(n(a),n(i(b,i(c,b)))),a)) | $ANS(step_14).
-P(i(i(n(a),n(i(i(b,c),i(i(c,d),i(b,d))))),a)) | $ANS(step_15).
-P(i(i(a,i(b,c)),i(a,i(i(c,d),i(b,d))))) | $ANS(step_16).
-P(i(i(a,b),i(n(b),i(a,c)))) | $ANS(step_17).
-P(i(i(i(a,b),c),i(n(a),c))) | $ANS(step_18).
-P(i(i(a,i(b,c)),i(b,i(a,c)))) | $ANS(step_19).
-P(i(i(a,b),i(i(i(b,a),a),b))) | $ANS(step_20).
-P(i(i(i(i(n(a),n(i(b,i(c,b)))),a),d),d)) | $ANS(step_21).
-P(i(a,i(i(b,c),i(i(a,b),c)))) | $ANS(step_22).
-P(i(i(i(n(a),i(b,c)),d),i(i(b,a),d))) | $ANS(step_23).
-P(i(n(n(a)),a)) | $ANS(lemma_24).
-P(i(i(a,b),i(i(c,a),i(c,b)))) | $ANS(lemma_25).
-P(i(i(a,i(n(b),n(i(c,i(d,c))))),i(a,b))) | $ANS(step_26).
-P(i(i(i(i(a,b),i(i(c,a),b)),d),i(c,d))) | $ANS(step_27).
-P(i(i(i(n(n(a)),b),c),i(i(a,b),c))) | $ANS(step_28).
-P(i(a,n(n(a)))) | $ANS(lemma_29).
-P(i(i(a,n(n(b))),i(a,b))) | $ANS(step_30).
-P(i(i(a,i(i(b,c),d)),i(a,i(c,d)))) | $ANS(step_31).
-P(i(i(a,i(n(b),n(c))),i(a,i(c,b)))) | $ANS(step_32).
-P(i(i(n(a),b),i(n(b),a))) | $ANS(step_33).
-P(i(i(n(n(a)),b),i(a,b))) | $ANS(step_34).
-P(i(i(i(a,b),n(c)),i(c,a))) | $ANS(step_35).
-P(i(i(a,b),i(n(b),n(a)))) | $ANS(lemma_36).
-P(i(i(a,i(n(n(b)),c)),i(a,i(b,c)))) | $ANS(step_37).
-P(i(i(a,i(b,c)),i(a,i(n(c),n(b))))) | $ANS(step_38).
-P(i(n(i(a,b)),n(b))) | $ANS(step_39).
-P(i(i(i(n(a),n(b)),c),i(i(b,a),c))) | $ANS(step_40).
-P(i(n(i(a,b)),n(i(i(b,c),n(a))))) | $ANS(step_41).
-P(i(i(i(a,b),i(c,d)),i(b,i(n(d),n(c))))) | $ANS(step_42).
-P(i(i(i(n(a),n(i(b,a))),n(i(b,a))),n(a))) | $ANS(step_43).
-P(i(i(a,i(i(n(b),n(c)),d)),i(a,i(i(c,b),d)))) | $ANS(step_44).
-P(i(i(i(n(a),n(b)),n(b)),i(i(a,b),n(a)))) | $ANS(step_45).
-P(i(i(a,n(i(b,c))),i(a,n(i(i(c,d),n(b)))))) | $ANS(step_46).
-P(i(i(a,i(i(n(b),n(i(c,b))),n(i(c,b)))),i(a,n(b)))) | $ANS(step_47).
-P(i(n(a),i(n(n(b)),n(i(b,a))))) | $ANS(step_48).
-P(i(i(i(i(a,b),b),n(i(a,b))),n(b))) | $ANS(step_49).
-P(i(n(a),i(b,n(i(b,a))))) | $ANS(step_50).
-P(i(i(a,i(i(i(b,c),c),n(i(b,c)))),i(a,n(c)))) | $ANS(step_51).
-P(i(i(i(n(a),i(b,n(i(b,a)))),c),c)) | $ANS(step_52).

-P(i(i(i(i(a,b),b),n(i(b,a))),n(a))) | $ANS(step_53).
-P(i(i(a,i(i(i(b,c),c),n(i(c,b)))),i(a,n(b)))) | $ANS(step_54).
-P(i(i(a,b),i(i(b,n(i(b,a))),n(a)))) | $ANS(step_55).
-P(i(i(i(a,n(i(a,b))),n(i(i(a,n(i(a,b))),n(b)))),n(n(b)))) | $ANS(step_56).
-P(i(i(i(a,n(i(a,b))),n(i(i(a,n(i(a,b))),n(b)))),b)) | $ANS(step_57).
-P(i(i(a,i(i(b,n(i(b,c))),n(i(i(b,n(i(b,c))),n(c))))),i(a,c))) | $ANS(step_58).
-P(i(i(i(a,n(i(a,b))),n(i(b,a))),b)) | $ANS(step_59).
-P(i(i(a,i(i(b,n(i(b,c))),n(i(c,b)))),i(a,c))) | $ANS(step_60).
-P(i(a,i(i(n(i(a,b)),n(i(b,a))),b))) | $ANS(step_61).
-P(i(a,i(i(i(b,a),i(a,b)),b))) | $ANS(step_62).
-P(i(i(i(a,b),i(b,a)),i(b,a))) | $ANS(step_63).
end_of_list.

Proof of MV5 in 63 Steps

1 [] -P(i(x,y)) | -P(x) | P(y).
2 [] P(i(x,i(y,x))).
3 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
4 [] P(i(i(i(x,y),y),i(i(y,x),x))).
5 [] P(i(i(n(x),n(y)),i(y,x))).
68 [] -P(i(i(i(a,b),i(b,a)),i(b,a))) | $ANS(step_63).
---
69 [hyper,1,2,2] P(i(x,i(y,i(z,y)))).
71 [hyper,1,3,3] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
73 [hyper,1,2,3] P(i(x,i(i(y,z),i(i(z,u),i(y,u))))).
75 [hyper,1,3,2] P(i(i(i(x,y),z),i(y,z))).
77 [hyper,1,3,4] P(i(i(i(i(x,y),y),z),i(i(i(y,x),x),z))).
79 [hyper,1,3,5] P(i(i(i(x,y),z),i(i(n(y),n(x)),z))).
81 [hyper,1,4,69] P(i(i(i(x,i(y,x)),z),z)).
83 [hyper,1,71,71] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
85 [hyper,1,71,3] P(i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))).
87 [hyper,1,4,73] P(i(i(i(i(x,y),i(i(y,z),i(x,z))),u),u)).
89 [hyper,1,75,5] P(i(n(x),i(x,y))).
91 [hyper,1,75,4] P(i(x,i(i(x,y),y))).
94 [hyper,1,77,77] P(i(i(i(x,i(y,x)),i(y,x)),i(i(i(x,y),y),x))).
98 [hyper,1,79,81] P(i(i(n(x),n(i(y,i(z,y)))),x)).
100 [hyper,1,79,87] P(i(i(n(x),n(i(i(y,z),i(i(z,u),i(y,u))))),x)).
102 [hyper,1,71,87] P(i(i(x,i(y,z)),i(x,i(i(z,u),i(y,u))))).
104 [hyper,1,83,89] P(i(i(x,y),i(n(y),i(x,z)))).
106 [hyper,1,3,89] P(i(i(i(x,y),z),i(n(x),z))).
108 [hyper,1,83,91] P(i(i(x,i(y,z)),i(y,i(x,z)))).
110 [hyper,1,75,94] P(i(i(x,y),i(i(i(y,x),x),y))).
112 [hyper,1,91,98] P(i(i(i(i(n(x),n(i(y,i(z,y)))),x),u),u)).
114 [hyper,1,102,91] P(i(x,i(i(y,z),i(i(x,y),z)))).
116 [hyper,1,3,104] P(i(i(i(n(x),i(y,z)),u),i(i(y,x),u))).
119 [hyper,1,106,100] P(i(n(n(x)),x)).
121 [hyper,1,87,108] P(i(i(x,y),i(i(z,x),i(z,y)))).
123 [hyper,1,71,112] P(i(i(x,i(n(y),n(i(z,i(u,z))))),i(x,y))).
125 [hyper,1,3,114] P(i(i(i(i(x,y),i(i(z,x),y)),u),i(z,u))).
127 [hyper,1,85,119] P(i(i(i(n(n(x)),y),z),i(i(x,y),z))).
129 [hyper,1,5,119] P(i(x,n(n(x)))).
137 [hyper,1,121,119] P(i(i(x,n(n(y))),i(x,y))).
140 [hyper,1,121,75] P(i(i(x,i(i(y,z),u)),i(x,i(z,u)))).
142 [hyper,1,121,5] P(i(i(x,i(n(y),n(z))),i(x,i(z,y)))).

144 [hyper,1,116,123] P(i(i(n(x),y),i(n(y),x))).
147 [hyper,1,3,129] P(i(i(n(n(x)),y),i(x,y))).
154 [hyper,1,142,106] P(i(i(i(x,y),n(z)),i(z,x))).
156 [hyper,1,127,144] P(i(i(x,y),i(n(y),n(x)))).
161 [hyper,1,121,147] P(i(i(x,i(n(n(y)),z)),i(x,i(y,z)))).
165 [hyper,1,121,156] P(i(i(x,i(y,z)),i(x,i(n(z),n(y))))).
167 [hyper,1,81,156] P(i(n(i(x,y)),n(y))).
169 [hyper,1,3,156] P(i(i(i(n(x),n(y)),z),i(i(y,x),z))).
171 [hyper,1,156,154] P(i(n(i(x,y)),n(i(i(y,z),n(x))))).
175 [hyper,1,140,165] P(i(i(i(x,y),i(z,u)),i(y,i(n(u),n(z))))).
177 [hyper,1,110,167] P(i(i(i(n(x),n(i(y,x))),n(i(y,x))),n(x))).
179 [hyper,1,121,169] P(i(i(x,i(i(n(y),n(z)),u)),i(x,i(i(z,y),u)))).
181 [hyper,1,77,169] P(i(i(i(n(x),n(y)),n(y)),i(i(x,y),n(x)))).
183 [hyper,1,121,171] P(i(i(x,n(i(y,z))),i(x,n(i(i(z,u),n(y))))).
186 [hyper,1,121,177] P(i(i(x,i(i(n(y),n(i(z,y))),n(i(z,y)))),i(x,n(y)))).
188 [hyper,1,175,181] P(i(n(x),i(n(n(y)),n(i(y,x))))).
191 [hyper,1,186,79] P(i(i(i(i(x,y),y),n(i(x,y))),n(y))).
193 [hyper,1,161,188] P(i(n(x),i(y,n(i(y,x))))).
195 [hyper,1,121,191] P(i(i(x,i(i(i(y,z),z),n(i(y,z)))),i(x,n(z)))).
197 [hyper,1,91,193] P(i(i(i(n(x),i(y,n(i(y,x)))),z),z)).
200 [hyper,1,195,77] P(i(i(i(i(x,y),y),n(i(y,x))),n(x))).
202 [hyper,1,121,200] P(i(i(x,i(i(i(y,z),z),n(i(z,y)))),i(x,n(y)))).
204 [hyper,1,125,202] P(i(i(x,y),i(i(y,n(i(y,x))),n(x)))).
207 [hyper,1,197,204] P(i(i(i(x,n(i(x,y))),n(i(i(x,n(i(x,y))),n(y)))),n(n(y)))).
210 [hyper,1,137,207] P(i(i(i(x,n(i(x,y))),n(i(i(x,n(i(x,y))),n(y)))),y)).
212 [hyper,1,121,210] P(i(i(x,i(i(y,n(i(y,z))),n(i(i(y,n(i(y,z))),n(z))))),i(x,z))).
214 [hyper,1,212,183] P(i(i(i(x,n(i(x,y))),n(i(y,x))),y)).
216 [hyper,1,121,214] P(i(i(x,i(i(y,n(i(y,z))),n(i(z,y)))),i(x,z))).
218 [hyper,1,125,216] P(i(x,i(i(n(i(x,y)),n(i(y,x))),y))).
220 [hyper,1,179,218] P(i(x,i(i(i(y,x),i(x,y)),y))).
222 [hyper,1,108,220] P(i(i(i(x,y),i(y,x)),i(y,x))).

Clause (222) contradicts clause (68), and the proof of *MV*5 is complete, with length 63 and level 23.

Because of our interest in using OTTER to seek elegant proofs—in this context, focusing on proof structure—we investigated the possibility of finding a proof of the theorem under discussion in which absent are all terms of the form $n(n(t))$ for any term t. From a respected colleague, we received the conjecture that such a proof did not exist. After all, experiments with many-valued sentential calculus and related calculi exhibited abundant use of the terms that we intended to avoid. At the same time, we planned to seek proofs shorter than 63 steps (of condensed detachment). After numerous experiments, we did in fact find (with OTTER) a 39-step proof free of any terms of the form $n(n(t))$. When we much later revisited the theorem in question, but added the use of the resonance strategy and dropped the term structure requirement, OTTER found a 35-step proof [Wos93b]. We close the history of this problem by noting that the knowledge of the 39-step proof with the term structure constraint and the 35-step proof without it caused us to study again (during the writing of this article) the theorem with the resonance strategy.

Our goal was to seek a proof that the four given axioms for many-valued sentential calculus imply the formula we call *MV*5 such that strictly less than 35 applications of condensed detachment are required and such that no term of the form $n(n(t))$ is present in any of the proof steps. In our first experiment, for resonators, we used correspondents of the four axioms (each with a weight of 1) that serve as the hypotheses of the theorem to be proved, followed by correspondents (32 of them each with a weight of 2) of the proof steps of the 35-step proof with the exception of those (the three) that contain a term of the form $n(n(t))$, followed by correspondents (15 of them each with a weight of 4) of the proof steps of the 39-step proof that are not present in the 35-step proof. All resonators were placed in weight_list(pick_and_purge). At the start of the run, the max_weight was set to 22; it was then reduced

to 16 after 1000 clauses were chosen as the focus of attention. A limit of 4 was placed on the number of distinct variables in a retained conclusion. The ratio strategy was used with the assigned parameter of 3. Ancestor subsumption was used, and so also was back subsumption. OTTER was instructed to drive its reasoning by first focusing on each of the four axioms, placed in list(sos), before choosing from the pool of deduced-and-retained clauses. Finally, the input for the first experiment included the following demodulator list, whose function we explain immediately.

```
list(demodulators).
(n(n(x)) = junk).
(i(i(x,x),y) = junk).
(i(y,i(x,x)) = junk).
(i(junk,x) = junk).
(i(x,junk) = junk).
(n(junk) = junk).
(P(junk) = $T).
end_of_list.
```

The first demodulator is included to begin the process of purging deduced clauses containing as a subterm a term of the form $n(n(t))$. The second and third demodulators are included to begin the process of purging deduced clauses that contain as a proper subterm a term of the form $i(t,t)$ for any term t. The remaining demodulators enable the program, with one exception, to complete the process of purging unwanted clauses of the types under discussion, for such clauses are eventually demodulated to $T, which the program treats as ''true'' and, therefore, purges. Regarding the exception, the given set of demodulators does *not* address the purging of a clause properly containing a term of the form $n(i(t,t))$ for some term t. One could certainly add two such demodulators, the following; we simply failed to do so.

```
(i(n(i(x,x)),y) = junk).
(i(y,n(i(x,x))) = junk).
```

In approximately 13 CPU-seconds on a SPARCstation 2, OTTER produced a 41-step proof of level 18 upon retention of a clause numbered 1823, and in approximately 353 CPU-seconds (in the same run) produced a 38-step proof of level 15 upon retention of a clause numbered 16228. The experiment had succeeded in one of its goals, that of producing a shorter proof (than previously known) of the theorem under study such that absent are the $n(n(t))$ terms. This success immediately led to the second experiment, that in which the second and third demodulators were commented out (by placing % in column 1). In other words, we permitted OTTER to retain clauses whether or not they contained as a proper subterm a term of the form $i(t,t)$.

In approximately 33 CPU-seconds on a SPARCstation 2, OTTER produced a 41-step proof of level 18 upon retention of a clause numbered 3264, and in approximately 409 CPU-seconds (in the same run) produced a 34-step proof of level 17 upon retention of a clause numbered 18192. A glance at the resulting 34-step proof (which we immediately give) produces a mystery (whose possible answer we give at the end of this subsection), for none of its steps takes advantage of the added latitude permitted in the second experiment. In fact, the 34-step proof contains but one clause that is not present in the 38-step proof found in the cited first experiment. In particular, none of the clauses contains as a proper subterm a term of the form $i(t,t)$.

A Nice 34-Step Proof with the Structure-Avoidance Property

```
1 [] -P(i(x,y))|-P(x)|P(y).
2 [] P(i(x,i(y,x))).
3 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
4 [] P(i(i(i(x,y),y),i(i(y,x),x))).
5 [] P(i(i(n(x),n(y)),i(y,x))).
6 [] -P(i(i(i(a,b),i(b,a)),i(b,a)))|$ANS(step_MV5).
----------------------------------------
```

17 [hyper,1,3,3] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
19 [hyper,1,3,2] P(i(i(i(x,y),z),i(y,z))).
20 [hyper,1,3,4] P(i(i(i(i(x,y),y),z),i(i(i(y,x),x),z))).
22 [hyper,1,3,5] P(i(i(i(x,y),z),i(i(n(y),n(x)),z))).
27 [hyper,1,17,17] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
32 [hyper,1,19,5] P(i(n(x),i(x,y))).
33 [hyper,1,19,4] P(i(x,i(i(x,y),y))).
38 [hyper,1,3,20] P(i(i(i(i(i(x,y),y),z),u),i(i(i(i(y,x),x),z),u))).
51 [hyper,1,3,22] P(i(i(i(i(n(x),n(y)),z),u),i(i(i(y,x),z),u))).
56 [hyper,1,22,4] P(i(i(n(x),n(i(y,x))),i(i(x,y),y))).
71 [hyper,1,27,32] P(i(i(x,y),i(n(y),i(x,z)))).
72 [hyper,1,3,32] P(i(i(i(x,y),z),i(n(x),z))).
75 [hyper,1,27,33] P(i(i(x,i(y,z)),i(y,i(x,z)))).
83 [hyper,1,33,32] P(i(i(i(n(x),i(x,y)),z),z)).
208 [hyper,1,22,75] P(i(i(n(i(x,y)),n(z)),i(x,i(z,y)))).
220 [hyper,1,75,3] P(i(i(x,y),i(i(z,x),i(z,y)))).
227 [hyper,1,27,220] P(i(i(x,i(y,z)),i(i(z,u),i(x,i(y,u))))).
254 [hyper,1,22,83] P(i(i(n(x),n(i(n(y),i(y,z)))),x)).
358 [hyper,1,227,75] P(i(i(i(x,y),z),i(i(x,i(u,y)),i(u,z)))).
360 [hyper,1,227,71] P(i(i(i(x,y),z),i(i(x,u),i(n(u),z)))).
361 [hyper,1,227,56] P(i(i(x,y),i(i(n(z),n(i(x,z))),i(i(z,x),y)))).
495 [hyper,1,360,254] P(i(i(n(x),y),i(n(y),x))).
537 [hyper,1,220,495] P(i(i(x,i(n(y),z)),i(x,i(n(z),y)))).
636 [hyper,1,537,72] P(i(i(i(x,y),z),i(n(z),x))).
648 [hyper,1,220,636] P(i(i(x,i(i(y,z),u)),i(x,i(n(u),y)))).
687 [hyper,1,636,5] P(i(n(i(x,y)),n(y))).
713 [hyper,1,33,687] P(i(i(i(i(n(i(x,y)),n(y)),z),z)).
728 [hyper,1,20,713] P(i(i(i(i(n(x),n(i(y,x))),n(i(y,x))),n(x))).
732 [hyper,1,51,728] P(i(i(i(i(i(x,y),y),n(i(x,y))),n(y))).
759 [hyper,1,38,732] P(i(i(i(i(i(x,y),y),n(i(y,x))),n(x))).
17438 [hyper,1,358,759] P(i(i(i(i(x,y),y),i(z,n(i(y,x)))),i(z,n(x)))).
17709 [hyper,1,648,361] P(i(i(x,y),i(n(i(i(z,x),y)),n(z)))).
17811 [hyper,1,17438,17709] P(i(n(i(i(i(x,y),i(y,x)),x)),n(y))).
18192 [hyper,1,208,17811] P(i(i(i(x,y),i(y,x)),i(y,x))).

Clause (18192) contradicts clause (6), and the proof of *MV*5 is complete, with length 34 and level 17.

The 34-step proof shares in common with the 35-step proof 29 steps. Three of the steps of the 34-step proof fail to match as a resonator any of the resonators corresponding to the steps of the 35-step proof. The 34-step proof shares in common with the 39-step proof 31 steps. Eight of the steps of the 34-step proof fail to match as a resonator any of those that correspond to the steps of the 39-step proof. Three steps of the 34-step proof are not among either the steps of the 35-step proof or the 39-step proof. Two of the steps of the 34-step proof fail to match as a resonator any resonator corresponding to a step in the 35-step proof or in the 39-step proof.

Regarding a possible answer to the mystery of why OTTER produced a 34-step proof in contrast to the 38-step proof it first found, we first comment on why it is in fact a mystery. First, to complete the 34-step proof, OTTER clearly does *not* follow the same paths it followed to produce the 38-step proof; otherwise, the program would in fact have found (in the first cited experiment) the given 34-step proof. Since only one change was made between the first and second experiments, the change accounts for following different paths. Second, because the change in the input gives OTTER more latitude, namely, the use of clauses properly containing terms of the form $i(t,t)$, one naturally conjectures that such clauses must play a key role. Third, one might therefore easily conjecture—we certainly did—that such clauses would appear in the 34-step proof, explaining the shortening by four steps of the desired proof in terms of the added latitude.

But, such clauses do *not* appear in the 34-step proof; indeed, the only clause in the 34-step proof that is not in the 38-step proof is the following.

P(i(n(i(x,y)),n(y))).

Clearly absent from the given clause are $i(t,t)$ terms. Except for $n(i(t,t))$ terms, the 38-step proof must satisfy the avoidance of $i(t,t)$ terms, for it is obtained with the first experiment, and that experiment prohibits their use. Inspection of the 38-step proof shows that such unwanted terms are absent, a fact that was also verified with an OTTER run that included two appropriately chosen demodulators.

The explanation of how OTTER was able to complete a 34-step proof—four steps shorter than the 38-step proof found in the first experiment—apparently rests with the data structures and the indexing of terms. Indeed, although they are not used in the desired proof, the retention of clauses properly containing $i(t,t)$ terms changes the contents of the data structure and affects the indexing of terms, which in turn causes the program to traverse different paths (through the search space of conclusions) than it otherwise would.

7.3. The Relevance of Elegance to Applications

Two applications are the focus of this subsection, logic circuit design and algorithm synthesis. Here we briefly discuss the relevance to these two applications of finding proofs with one or more of the three properties of elegance that have been under discussion. In doing so, we show how the resonance strategy is useful in solving what might be termed practical problems. We begin with proof length.

When the problem is one either of circuit design or algorithm synthesis, one can profitably use the ANSWER literal offered by OTTER. In particular, depending on the nature of the proof, if one is found and the argument of the ANSWER literal is suitably prepared, the ANSWER literal will contain as its argument the corresponding circuit that is designed or algorithm that is synthesized. In Chapter 7 of [Wos92], we give various examples of using OTTER to design circuits, including the following rather well known two-inverter circuit-design problem.

> Using as many AND and OR gates as you like, but using only two NOT gates, can you design a circuit according to the following specification? There are three inputs, i1, i2, and i3, and three outputs, o1, o2, and o3. The outputs are related to the inputs in the following simple way: o1 = not(i1) o2 = not(i2) o3 = not(i3).

Often, the shorter the proof, the fewer components in the constructed object. Everything being equal, the fewer the gates a circuit requires, the more attractive is the circuit. A similar remark holds for algorithm synthesis. We note that we have experimented very little with that application. If we now take into account the use of the resonance strategy in successfully seeking shorter proofs, we see how this strategy is relevant to the two applications under discussion.

Regarding the second aspect of elegance, term structure, the cited two-inverter problem provides an excellent example. The problem requires using no more than two NOT gates. When OTTER is asked to construct the desired circuit, the corresponding proof that is sought is constrained in that no step is allowed to contain more than two occurrences of the function *not*. As for algorithm synthesis, one can easily imagine the case in which one wishes to avoid nested *divide* instructions. As discussed earlier in this section, the resonance strategy can be put to good use in this context when coupled with demodulators of the type given in Section 7.2.

As for the third aspect of elegance cited in this article, that of compactness, its presence in a proof can signal a breakthrough in efficiency either for circuit design or for algorithm synthesis. Indeed, imagine that the goal is the design of three circuits and that the assignment is given to OTTER by placing in list(passive) what amounts to the assumption that such is not possible for any of the three. The situation is like that in which the object is to prove each of theses 18, 35, and 49 (the Church axiom system) by using the Lukasiewicz axiom system consisting of $L1$, $L2$, and $L3$. In place of the Lukasiewicz axioms, let us assume that the input includes axioms that give the properties of the components to be used, for example, the properties of various gates. If OTTER is successful in finding the

desired three proofs, if the use of the ANSWER literal results in the presentation of each of the sought-after circuits, and if one of the proofs is compact, then the other two proofs are subproofs of the compact proof. Should such occur, one has been presented with three circuits of interest, two of which are subcircuits of the third. One might then be able to take advantage of the given design to save on components and, with suitable actions, to add to the efficiency of the global design. Although we have had virtually no experience with algorithm synthesis, similar observations hold for algorithm synthesis in the context of finding a compact proof. Such a proof can provide one with code for one subroutine that has within it the needed appropriate code for smaller subroutines, reminiscent of what compilers attempt to produce to increase program efficiency. Because the resonance strategy can be used to squeeze steps out of a proof (by using it to seek shorter proofs), this strategy may prove useful when the goal is that of increasing efficiency either for circuit design or for algorithm synthesis.

8. Summary and Conclusions

In this article, we define the resonance strategy and present evidence of its value, particularly in the context of seeking more elegant proofs. The resonance strategy asks the researcher to choose resonators, formulas and equations whose functional shape (ignoring the particular variable occurrences within them) suggests they will be useful in directing an automating reasoning program's search for assignment completion. To each resonator, one assigns an integer, called its value or weight, that prioritizes similar expressions: the smaller the integer, the higher the priority for being chosen to direct the reasoning. An expression is similar to a given resonator if and only if the two are identical when the variables are ignored, in other words, the two have the same shape or functional skeleton.

In one sense, resonators generalize lemmas. For example, where in Boolean groups—those in which the square of every x is the identity element e—the lemmas $yzyz = e$ and $yyzz = e$ are such that neither generalizes the other, the resonator $xyzx = e$, by treating variables as indistinguishable, generalizes and captures (in a manner that is discussed in this article) both lemmas. Where resonators (which do *not* take on the value true or false) are used to guide a reasoning program's search, lemmas are basic truths from which to reason.

One of the more successful uses of the resonance strategy is that of finding shorter proofs; in this article, we give examples from group theory, Robbins algebra, and various logic calculi. From the perspective of science as a whole, the successful search for shorter proofs can result in the discovery of useful lemmas that are fundamental to the field under study. The use of lemmas (or basic truths) not only often is the means for reducing the length of a proof and adding elegance but frequently leads to significant advances such as the design of a more efficient circuit or a more effective algorithm. Therefore, one benefit of using the resonance strategy often is that of finding useful and sometimes powerful lemmas.

McCune's automated reasoning program OTTER offers a mechanism, weighting, that provides the researcher a convenient way to rely on the resonance strategy. However, since the resonance strategy is used to direct a program's search by giving preference to formulas or equations that match at the skeletal level (ignoring variables) some resonator, the strategy can be implemented with a string-comparing algorithm rather than with weighting. With OTTER, one simply places the chosen resonators with the chosen value in the appropriate weight_list, usually in weight_list(pick_and_purge). A judicious choice of resonators can transform a question to be answered from the unreachable to the easy-to-answer. Intuitively, the explanation rests with the fact that the resonance strategy provides a means for escaping from cul-de-sacs, navigating potentially wide valleys, and crossing high plateaus. By a wide valley, we refer to the case in which one needs the use of a conclusion with, say, 18 symbols in it and one is mired in a huge space of conclusions each consisting of 14 symbols. By a high plateau, we refer to the case in which the proof requires the use of three or more consecutive conclusions each of whose complexity is at least moderately greater than the vast majority of conclusions being retained. Clearly, the resonance strategy is but one piece for solving the puzzle of proof finding, whether or not the assignment includes the search for *short* proofs.

References

[Kunen93] Kunen, K., Private communication (1993).

[Lukasiewicz63] Lukasiewicz, J., *Elements of Mathematical Logic,* Pergamon Press, Oxford (1963).

[Lukasiewicz70] Lukasiewicz J., *Selected Works,* ed. L. Borkowski, North-Holland, Amsterdam (1970).

[McCharen76] McCharen, J., Overbeek, R., and Wos, L., ''Complexity and related enhancements for automated theorem-proving programs'', *Computers and Mathematics with Applications,* 2, pp. 1-16 (1976).

[McCune90] McCune, W., *OTTER 2.0 Users Guide,* Technical Report ANL-90/9, Argonne National Laboratory, Argonne, Illinois (1990).

[McCune93] McCune, W., ''Single axioms for groups and Abelian groups with various operations'', *J. Automated Reasoning,* 10, no. 1, pp. 1-13 (January 1993).

[Scott90] Scott, D., Private communication (1990).

[Slaney92] Slaney, J., *FINDER: Finite Domain Enumerator, Version 2.0, Notes and Guide,* Technical Report, Australian National University, 1992.

[Winker90] Winker, S., ''Robbins algebra: Conditions that make a near-Boolean algebra Boolean'', *J. Automated Reasoning,* 6, no. 4, pp. 465-489 (December 1990).

[Winker92] Winker, S., ''Absorption and idempotency criteria for a problem in near-Boolean algebras'', *J. Algebra,* 153, no. 2, pp. 414-423 (December 1992).

[Wos65] Wos, L., Robinson, G., and Carson, D., ''Efficiency and completeness of the set of support strategy in theorem proving'', *J. of the ACM,* 12, pp. 536-541 (1965).

[Wos87] Wos, L., *Automated Reasoning: 33 Basic Research Problems,* Prentice-Hall: Englewood Cliffs, New Jersey (1987).

[Wos91a] Wos, L., and McCune, W., *The Application of Automated Reasoning to Proof Translation and to Finding Proofs with Specified Properties: A Case Study in Many-Valued Sentential Calculus,* Technical Report ANL-91/19, Argonne National Laboratory, Argonne, Illinois (August 1991).

[Wos91b] Wos, L., ''Automated reasoning and Bledsoe's dream for the field,'' pp. 297-345 in *Automated Reasoning: Essays in Honor of Woody Bledsoe,* ed. R. S. Boyer, Kluwer Academic Publishers, Dordrecht (1991).

[Wos92] Wos, L., Overbeek, R., Lusk, E., and Boyle, J., *Automated Reasoning: Introduction and Applications,* 2nd ed., McGraw-Hill, New York (1992).

[Wos93a] Wos, L., ''Automated reasoning answers open questions,'' *Notices of the AMS,* 5, no. 1, pp. 15-26 (January 1993).

[Wos93b] Wos, L., unpublished information, Argonne National Laboratory (1993).